DOCTOR OF ENGINEERING (ENGD)

Meta-Parametric Design

Developing a Computational Approach for Early Stage Collaborative Practice

Harding, John

Award date:
2015

Awarding institution:
University of Bath

Meta-Parametric Design

Developing a Computational Approach for Early Stage Collaborative Practice

A thesis submitted for the degree of

Doctor of Engineering

John Harding

University of Bath
Department of Architecture and Civil Engineering

July 2014

Abstract

Computational design is the study of how programmable computers can be integrated into the process of design. It is not simply the use of pre-compiled computer aided design software that aims to replicate the drawing board, but rather the development of computer algorithms as an integral part of the design process. Programmable machines have begun to challenge traditional modes of thinking in architecture and engineering, placing further emphasis on *process* ahead of the final result. Just as Darwin and Wallace had to think beyond form and inquire into the *development* of biological organisms to understand evolution, so computational methods enable us to rethink how we approach the design process itself.

The subject is broad and multidisciplinary, with influences from design, computer science, mathematics, biology and engineering. This thesis begins similarly wide in its scope, addressing both the technological aspects of computational design and its application on several case study projects in professional practice. By learning through participant observation in combination with secondary research, it is found that design teams can be most effective at the early stage of projects by engaging with the additional complexity this entails.

At this concept stage, computational tools such as parametric models are found to have insufficient flexibility for wide design exploration. In response, an approach called Meta-Parametric Design is proposed, inspired by developments in genetic programming (GP). By moving to a higher level of abstraction as computational designers, a Meta-Parametric approach is able to adapt to changing constraints and requirements whilst maintaining an explicit record of process for collaborative working.

Acknowledgements

This thesis is dedicated to my wife Simone and our daughter Annabel. I am forever grateful.

Thank you to my three supervisors Paul Shepherd, Chris Williams and Duncan Horswill for their expert guidance. I distinctly remember Paul calling me just before I committed to ask "are you absolutely sure you want to do this?!" I forget my reply, but I presume he must have convinced me to say yes and I'm pleased that he did.

Thank you to my friends and colleagues at Ramboll, especially Stephen Melville, Harri Lewis, Andreas Bak, Alex Baalham, Iain Sproat, Tom Foley, Ross Smith, Mark Pniewski, Yanchee Lau, Will Pearson and Kristjan Plagborg Nielsen.

Thank you to those in the computational design community, in particular Sam Conrad Joyce, Robert Aish, Daniel Piker, Gennaro Senatore, Emmanouil Zaroukas, Christian Derix and the late Paul Coates. Paul was an inspiration to all of his former students and we hope his spirit lives on through our work.

Thank you to all the collaborators outside of Ramboll that have given up their time to participate in this research. Particular mention to Charlie Whitaker (3DReid), Daniel Nielsen & Brian Sheldon (AG5 Architects), Jim Dodson (SPINN Arkitekter), Chun Qing Li (Pavilion Architecture) and Catherine Huang (Bjarke Ingels Group) for allowing me to participate on various influential projects that helped shape this work.

Thank you to the Systems Centre at The University of Bristol & The University of Bath, the Engineering and Physical Sciences Research Council and Ramboll UK for jointly supporting this research. Without the EngD programme, my initial meanderings to discover something I didn't know I was looking for would never have happened.

Finally, thank you to my family for everything else.

Contents

I. Introduction **1**

1. Introduction **3**
1.1. EngD Beginnings 4
1.1.1. Ramboll and I 4
1.1.2. Ramboll's Initial Intentions 5
1.1.3. What is Computational Design? 6
1.2. Thesis Objective 8
1.2.1. Computational Design 8
1.2.2. Meta-Parametric Design 9
1.3. Methodology 10
1.3.1. Data Triangulation 10
1.3.2. Primary Research Methods 11
1.3.3. Secondary Research Methods 13
1.3.4. An Adaptive Methodology 13
1.4. Contribution to Knowledge 14
1.5. Chapter Outline 15

II. Review **17**

2. Post-Rationalisation **19**
2.1. Introduction 19
2.1.1. Building Models 20
2.2. Heuristic Algorithms 20
2.2.1. Introduction 20
2.2.2. Without External Analysis 21
2.2.3. Consistent External Analysis 26
2.2.4. Incremental External Analysis 30
2.2.5. Summary 35
2.3. Metaheuristic Search Algorithms 36
2.3.1. Introduction 36
2.3.2. Metaheuristic Examples 37
2.3.3. Mapping Process 40
2.4. Parametric Modelling 41
2.4.1. Dataflow Programming 42
2.4.2. Integrating with Building Analysis 43
2.4.3. Integrating with Metaheuristics 45
2.5. Multi-Objective Optimisation 47
2.5.1. Adapting the Parametric Model 48
2.5.2. Handling Requirement Changes 49
2.6. Discussion 51
2.6.1. Engineer as Problem Solver 51
2.6.2. Review of Aims & Objectives 52

3. Pre-Rationalisation **55**
3.1. Introduction 55

3.2. Engineering Led Design . . . 56
3.2.1. Interactive Form-Finding . . . 56
3.2.2. Qualitative Assessment . . . 59
3.2.3. Case Study Results . . . 60
3.2.4. Summary . . . 62
3.3. Introducing Stakeholders . . . 63
3.3.1. Air Baltic Terminal Competition . . . 63
3.3.2. The KREOD Pavilion . . . 65
3.3.3. Summary . . . 66
3.4. Discussion . . . 67
3.4.1. Primary Generators . . . 67
3.4.2. Working Together . . . 68
3.4.3. Concept Design Stage . . . 68
3.4.4. Review of Aims & Objectives . . . 70

III. Working Together **73**

4. Early Parametric Design **75**
4.1. Introduction . . . 75
4.2. Why Parametric Models? . . . 75
4.2.1. Working with the Graph . . . 76
4.2.2. A Collaborative Framework . . . 77
4.3. Design Exploration . . . 79
4.4. Implementation . . . 80
4.4.1. Constraints and Objectives . . . 81
4.4.2. Performance Evaluation . . . 82
4.5. Observations . . . 82
4.5.1. Concept Design Exploration . . . 84
4.5.2. A Fixed Body-Plan . . . 85
4.5.3. Design development . . . 86
4.5.4. Possible Solutions . . . 88
4.6. Discussion . . . 89
4.6.1. Relation to Software Development . . . 89
4.6.2. Review of Aims and Objectives . . . 90

5. Freeing the Body Plan **93**
5.1. Introduction . . . 93
5.1.1. Evolutionary Development . . . 94
5.2. Artificial Embryogenesis . . . 95
5.2.1. Local Rules . . . 95
5.3. Implementation . . . 96
5.3.1. Modelling . . . 97
5.3.2. Local Rules . . . 97
5.3.3. Performance Objectives . . . 99
5.3.4. User Control . . . 100
5.3.5. Using a Metaheuristic . . . 101
5.4. Observations . . . 101
5.4.1. Collaboration Issues . . . 103
5.4.2. Case Study Evidence . . . 105
5.4.3. Evidence from Industry . . . 107
5.5. Discussion . . . 108
5.5.1. Reconsidering Parametric Models . . . 109
5.5.2. Evolvability . . . 110
5.5.3. Review of Aims and Objectives . . . 111

IV. Bridging The Gap 113

6. Meta-Parametric Design 115
6.1. Introduction . . . 115
6.2. Genetic Programming . . . 116
6.2.1. Tree Representation . . . 116
6.2.2. Generative Grammars . . . 118
6.2.3. GP for Design Exploration . . . 119
6.2.4. Cartesian Genetic Programming . . . 120
6.3. Embryo . . . 122
6.3.1. Introduction . . . 122
6.3.2. Generating DAGs . . . 123
6.3.3. Embryo Components . . . 125
6.3.4. Constraints and Parameters . . . 125
6.3.5. Encoding and Generation . . . 127
6.3.6. Some Simple Examples . . . 130
6.3.7. Combining with Metaheuristics . . . 133
6.3.8. Cognition . . . 134
6.3.9. Working In-Between . . . 136
6.4. Discussion . . . 137

7. Embryo Experiments 139
7.1. Experiment 1: AG5 Architects . . . 139
7.1.1. Set up . . . 139
7.1.2. Modelling . . . 141
7.1.3. Model Examples . . . 143
7.1.4. Performance Evaluation . . . 143
7.1.5. Model Cognition . . . 147
7.1.6. General Observations . . . 148
7.1.7. Improvements . . . 150
7.2. Experiment 2: 3DReid Architects . . . 152
7.2.1. Set up . . . 152
7.2.2. Modelling . . . 153
7.2.3. Design Exploration . . . 154
7.2.4. Graph Complexity as an Objective . . . 156
7.2.5. Post-Generation . . . 157
7.3. Experiment 3: Shape Analysis . . . 160
7.3.1. A Suitable Metaheuristic . . . 160
7.3.2. Results . . . 161
7.3.3. Increasing Complexity . . . 161
7.3.4. Opening Dead-ends . . . 165
7.3.5. Forget about Legibility . . . 167
7.4. Discussion . . . 169

V. Conclusions 171

8. Conclusions 173
8.1. Meta-Parametric . . . 173
8.1.1. Summary . . . 173
8.2. Embryo Results . . . 174
8.2.1. Model Design Space . . . 175
8.2.2. Model Performance . . . 176
8.2.3. Graph Intelligibility . . . 176
8.3. Further work . . . 177
8.3.1. Better Cognition tools . . . 177

8.3.2. Recursive & Cyclic Structures . . . 177
8.3.3. Hierarchies of Scale . . . 178
8.3.4. Artificial Selection . . . 178
8.4. Final Comment . . . 180

VI. Appendix **199**

A. Case Study Details **201**

B. Principal Stress Trajectories **235**
B.1. Introduction . . . 236
B.2. Real-time Software . . . 236
B.3. Calculation Method . . . 237
B.4. Principal Bending Stress . . . 238
B.5. Conclusion . . . 240

C. Structural Form-finding using Springs and Dynamic Weights **241**
C.1. Background . . . 241
C.1.1. Zero-length springs . . . 242
C.2. Two-dimensional systems . . . 242
C.2.1. Method overview . . . 242
C.3. Extension to funicular forms . . . 243
C.3.1. Dynamic Weights . . . 243
C.3.2. Real-time exploration . . . 244
C.3.3. Catenary Comparison . . . 244
C.4. Three-dimensional systems . . . 245
C.4.1. Using a Trivalent Topology . . . 246

D. The London Foyer Sculpture **249**
D.1. Introduction . . . 249
D.2. Design Strategy . . . 249
D.3. Material & Fabrication . . . 250
D.4. Connection Details . . . 250
D.5. Assembly . . . 251

E. The TRADA Pavilion – A Timber Plate Funicular Shell **253**
E.1. Introduction . . . 253
E.2. Funicular Form-finding . . . 254
E.2.1. Computational Approach . . . 254
E.2.2. Zero-Length Spring System . . . 255
E.2.3. Software Application . . . 256
E.3. Planar Re-meshing . . . 256
E.3.1. Triangles & Quads . . . 257
E.3.2. Tri-valent Planar Polygons . . . 259
E.3.3. Free Edges . . . 259
E.4. Fabrication & Assembly . . . 260
E.4.1. Connection Detail . . . 261
E.4.2. Assembly . . . 261
E.5. Conclusion . . . 262

F. Selected Algorithms **265**

G. Embryo Examples **271**

H. More Embryo Examples **275**
H.1. Tower Hamlets: Embryo Setup . . . 275

H.2. Tower Hamlets: Examples 275
H.3. Tower Hamlets: Graph Edit 275

List of Figures

1.1. Location within industry hierarchy 5
1.2. Building model performance criteria 6
1.3. Question hierarchy within wider context 9
1.4. Case study project involvement 12

2.1. Post-rationalising an existing design 21
2.2. Garden Festival minimal surface 22
2.3. Laplacian smoothing on the Lusail Bridge Roof 23
2.4. KREOD surface relaxation 24
2.5. Architectural concept for The Astana National Library . . 24
2.6. Initial facade triangulation for Astana National Library . . 25
2.7. Astana National Library nodal repulsion 26
2.8. Algorithm with external analysis 26
2.9. PQ meshing process 27
2.10. Senate House glazed roof 27
2.11. Orchid House discretisation 28
2.12. Helsinki Central Library: principal bending stress slab . . 29
2.13. Passive solar design Java application 30
2.14. Majlevik Bay Villas: generated roof forms 30
2.15. Algorithm using dynamic external analysis 31
2.16. Node pushing to high stress 31
2.17. Astana Library: nodal displacements 32
2.18. Astana Library node pushing 33
2.19. Tallinn Town Hall topology optimisation 34
2.20. Metaheuristic workflow 37
2.21. Bath House column optimisation 38
2.22. Metaheuristic solver comparison 39
2.23. The process of genotype manipulation for a typical GA . . 39
2.24. Citylife Tower Spire evolution 40
2.25. Direct mapping from genotype to point location 41
2.26. Probably the simplest parametric model ever made 42
2.27. Exploring façade densities on Arts Alliance 43
2.28. Cheongna Tower real-time structural analysis 44
2.29. Parametric model workflow 45
2.30. Cheonga Tower optimisation 46
2.31. A simple multi-objective problem 48
2.32. Multi-objective optimisation on Cheongna Tower 49
2.33. Cheongna Tower: structural analysis and visualisation . . 50
2.34. Traditional procurement 51

3.1. Design exploration with a heuristic method 56
3.2. The Ongreening Pavilion realised in March 2014 57
3.3. Mannheim Multihalle hanging chain model 58
3.4. TRADA Pavilion form-finding 59
3.5. A qualitative requirement for TRADA 60
3.6. TRADA Pavilion design exploration 60
3.7. Completed TRADA Pavilion 61
3.8. Two computational heuristics run sequentially 61

3.9. Node detail on the London Foyer Sculpture 62
3.10. Embedding engineering heuristics in software 63
3.11. Air Baltic Terminal: initial roof geometry 64
3.12. Java application used on the Air Baltic Terminal 64
3.13. Air Baltic Terminal: structural analysis 65
3.14. KREOD: Surface principal curvature for five options 66
3.15. Unloved ESS competition shells 67
3.16. Freedom vs knowledge plot 69

4.1. Spaghetti 77
4.2. Collaborative model used on the Aviva Stadium Project . . 78
4.3. Escher tower foam models 81
4.4. Constraints and objectives search 82
4.5. Escher tower performance feedback 83
4.6. Karl Sims' evolving creatures 86
4.7. Different graph definitions leading to an identical form . . 87
4.8. Design process and building model development 90

5.1. The influence of Hox genes on the body-plan 94
5.2. Graph representation of local buildings 97
5.3. ENI HQ foam models 98
5.4. Development of form using the ENI graph generator . . . 99
5.5. Screen-shots from the Java application 100
5.6. Brute-force search of outputs with two objectives 101
5.7. A sample of good designs close to the Pareto front 102
5.8. Pluit City cellular automaton 103
5.9. Geometric 'gestures' by BIG 106
5.10. Explicit and implicit embryogenies 109
5.11. Dawkin's Biomorphs 110

6.1. A soap film finds a minimal surface 116
6.2. Genetic programming with LISP trees 117
6.3. Senatore's Tower Generator 119
6.4. Compositional Pattern Producing Network examples . . . 120
6.5. Tree vs DAG 121
6.6. A Cartesian Genetic Programming example 122
6.7. A typical grasshopper component 123
6.8. Typical Embryo set up 124
6.9. Embryo component list 126
6.10. The main Embryo Component 128
6.11. Graph generation process 129
6.12. Three generated graph structures 131
6.13. Wide range of phenotypes generated by Embryo 132
6.14. Junk components 133
6.15. DAG and Embryo 134
6.16. Embryo generates topologically sorted models 136
6.17. Layered model showing model hierarchy 137

7.1. Formulating constraints and objectives collaboratively . . 140
7.2. Embryo set up on Gran Rubina Tower 142
7.3. Tower phenotypes A to D 144
7.4. Tower graph definitions A to C 145
7.5. Relative performance results for various tower typologies 146
7.6. Ranking tower designs according to performance 146
7.7. Pniewski modifications to design 'B' 149
7.8. Option D graph cleaned 151
7.9. Tower Hamlets grammar 153

7.10. Putra Heights, Malaysia 153
7.11. Two massing studies for Tower Hamlets 154
7.12. Graph and phenotype for Design 'A' 155
7.13. Manually adjusting Option 'D' post creation 156
7.14. Graph from Design 'A' manually untangled 156
7.15. Taking an Embryo design forward 158
7.16. Graph development 159
7.17. Embryo shape analysis with Simulated Annealing 162
7.18. Generated parametric definition for target geometry 1 . . . 162
7.19. Embryo finds the target 2 shape 164
7.20. Generated parametric definition for target geometry 2 . . . 164
7.21. Same geometry - New graph 166
7.22. Evolved graph 167
7.23. Alternative solution space 167
7.24. An unintelligible graph 168

8.1. GL-System 178
8.2. William Latham: generative art 179

B.1. Vikki Synergy Building principal stress ribs 235
B.2. Wolff's sketch of the femur showing material alignment . . 236
B.3. Principal stress real-time solver 237
B.4. Spring finite element 237
B.5. Nervi slab comparison 238
B.6. Ribbed slab proposal for the Ramboll UK foyer space . . . 239

C.1. Zero-length springs form a parabola with equal masses . . 243
C.2. Exploring catenaries by varying the gravitational field . . . 244
C.3. 17 node system compared to an actual catenary 245
C.4. Increasing of nodes reduces error to zero 245
C.5. Using a trivalent spring topology 247

D.1. Funicular form-finding approach 249
D.2. Members set out following surface principal curvature field 250
D.3. Foyer sculpture connection detail 251
D.4. Final constructed gridshell 251

E.1. Shell form-finding using the dynamic weights 256
E.2. Final fine triangulated mesh sent for re-meshing 257
E.3. The funicular surface discretised with triangular plates . . 258
E.4. The three-plate principle 258
E.5. TRADA surface discretised 260
E.6. TRADA: edge stiffener analysis 260
E.7. Identical hinges were used to connect every panel 261
E.8. Assembled node detail 262
E.9. Completed Shell at the Surface Design Awards (2013) . . . 263
E.10. Completed Shell at the UK Timber Expo (2012) 263

G.1. Different examples for 8 sliders and 16 components 272
G.2. Different examples for 16 sliders and 32 components 273
G.3. Examples for 80 sliders and 100 components 274

H.1. Tower Hamlets setup 276
H.2. Design 'B' 277
H.3. Design 'C' 278
H.4. Design 'D' 279
H.5. Design 'E' 280

H.6. Design 'F' 281
H.7. Graph Development post-Embryo (1/2) 282
H.8. Graph Development post-Embryo (2/2) 283

List of Case Studies

The following list contains 33 selected projects that I participated in during my time at Ramboll Computational Design. Although most of the projects are collaborative, the work presented in the thesis is the author's work unless stated otherwise. Each project has a short a name reference that is used throughout this thesis. A detailed description of each project can be found in Appendix A.

1. Astana National Library ..... [01Anl]
2. Lusail Bridge Roof ..... [02Lbr]
3. Cheongna Tower ..... [03Cht]
4. Bath House Column Optimisation ..... [04Bhc]
5. Arts Alliance Travelling Theatre ..... [05Aat]
6. National Holdings Headquarters ..... [06Nhq]
7. London Foyer Sculpture ..... [07Foy]
8. Urban Bubble Gridshell ..... [08Urb]
9. National Gallery of Greenland ..... [09Nuk]
10. Viikki Synergy Building ..... [10Vik]
11. Garden Festival Pavilion ..... [11Gfp]
12. KREOD Pavilion ..... [12Kre]
13. Air Baltic Terminal Competition ..... [13Rig]
14. Markham Vale Tower ..... [14Mvt]
15. Tallinn Town Hall ..... [15Tth]
16. Escher Tower ..... [16Esc]
17. Forgotten Spaces Competition ..... [17Fsc]

18. RIBA Pylon Competition ..... [18Pyl]

19. Maljevik Bay Resort ..... [19Mjb]

20. ENI Exploration and Production HQ ..... [20Eni]

21. Cockaigne House Shell ..... [21Chs]

22. Orchid House ..... [22Orc]

23. Senate House Roof ..... [23Shr]

24. TRADA Pavilion ..... [24Tra]

25. Vestas Blade Technology Centre ..... [25Ves]

26. Helsinki Central Library ..... [26Hcl]

27. Citylife Tower Spire ..... [27Cts]

28. York Way Apartments ..... [28Yor]

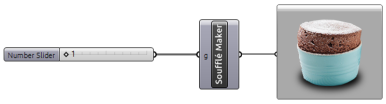
29. Gran Rubina Tower 3 ..... [29Grt]

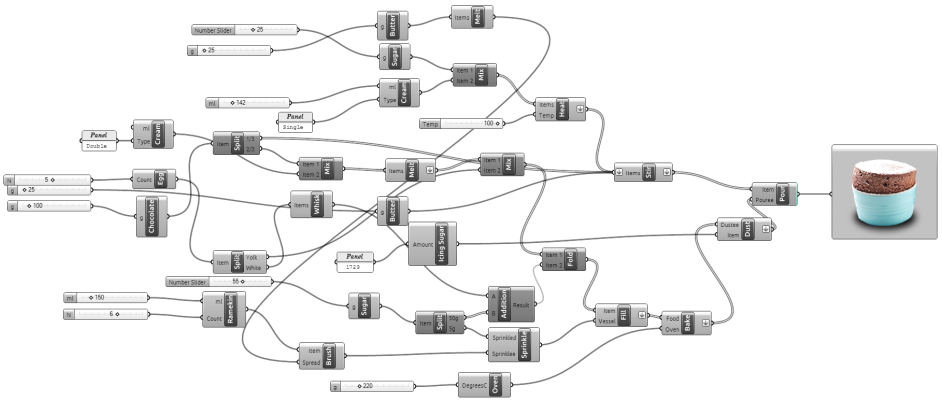
30. European Spallation Source Building ..... [30Ess]

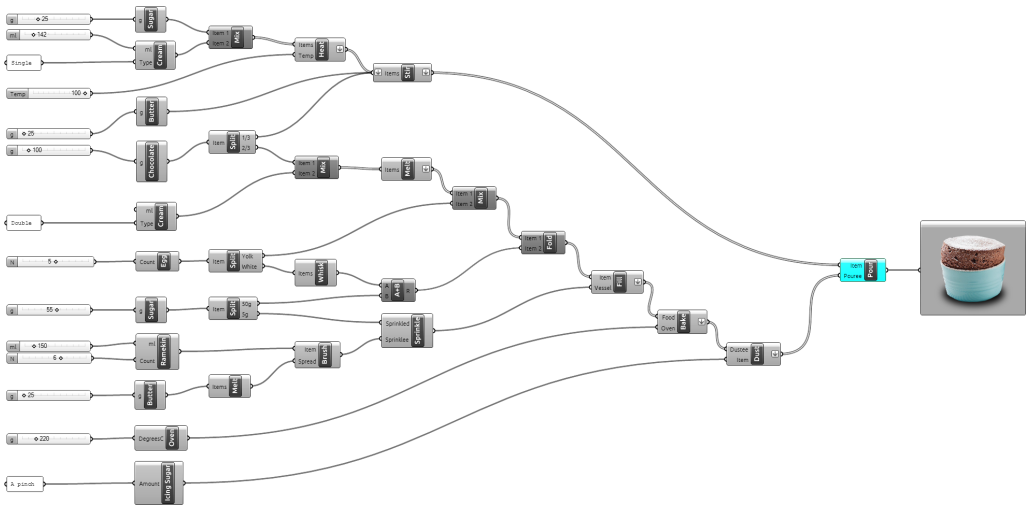
31. Tivoli Edge Development ..... [31Tiv]

32. Pluit City ..... [32Plu]

33. Tower Hamlets Apartments ..... [33Tow]


Number Slider
1
Soufflé Maker


Number Slider
25
Butter
Sugar
Items
Make
Item 1
Item 2
Mix
Type
Create
142
Panel
Single
Temp
100
Items
Temp
Make
Panel
Double
Split
Items
Mix
Items
Mix
Items
Slice
Item
Pour
Count
Egg
Chocolate
Items
Whisk
Butter
Amount
Icing Sugar
Duisse
Item
Dust
Item
Split
Yolk
White
Panel
1729
Item 1
Item 2
Fold
Number Slider
55
Ramekin
Count
Butter
Split
Item
50g
SQ
Arrange
Result
Sprinkled
Sprinkles
Sprinkle
Item
Vessel
Fill
Food
Oven
Bake
Item
Spread
Butter
Degrees C
Oven
220


Sugar
25
142
Single
Type
Create
Items
Temp
Make
Items
Slice
Temp
100
Butter
25
100
Chocolate
Item
Split
1/3
2/3
Items
Mix
Items
Mix
Item 1
Item 2
Mix
Item 1
Item 2
Fold
Double
Type
Create
Count
Egg
5
Item
Split
Yolk
White
Whisk
Arrange
A
B
Item
Vessel
Fill
Food
Oven
Bake
Duisse
Item
Dust
Item
Pour
Sugar
Item
Split
50g
SQ
55
Sprinkled
Sprinkles
Sprinkle
Ramekin
Count
Item
Spread
Butter
150
6
Butter
Items
Make
25
Degrees C
Oven
220
Amount
Icing Sugar
A pinch

Part I.

Introduction

1. Introduction

> "Whilst this planet has gone cycling on according to the fixed law of gravity, from so simple a beginning endless forms most beautiful and most wonderful have been, and are being, evolved."
>
> Charles Darwin, The Origin of Species,1872

When we think of architecture, images of pristine buildings often published in a glossy magazine or website spring to mind. Whether these are unrealised ideas or constructed projects, the observer from the outside usually has access only to the finished article with any messy details of how it got there in the first place obscured from view. On the inside however lies a narrative that enables a deeper understanding of the work - the struggle of *how and why* it came to be. It is this process of how the building came into being that enables others to learn about the design methods used, the intention of the work and how any collaboration between individuals was actually achieved (if at all).

Similarly, this is probably the first part of the thesis you are reading but the last part I am actually writing. Of course you wouldn't have known that had I not just told you (and indeed you may not really care), but it might be useful to someone thinking about writing a thesis themselves. The process of writing this thesis is somewhat similar to the content it attempts to describe; not an investigation of a single idea, but the *process* of gradually arriving at a final result through a series of insights and decisions. Here we may use a natural analogy, for it is only by considering the *process* of evolutionary development that one can begin to understand how such "endless forms most beautiful" as Darwin put it have come to exist today.

In the evolution and development of natural organisms we see a direct comparison to computation, as manipulation takes place at the level of process in the form of a program. This nature of computing is enacting new ways of thinking about building design, especially within a collaborative environment. Writing computer programs to perform well defined tasks is actually relatively easy when compared to the

challenge of interaction with multiple humans for creating and solving design problems. Firstly, one is presented with changing people's existing working patterns so that new methods can become adopted. The human aspect of integrating computers into the creative process of collaborative practice is therefore of central importance to this thesis. Computational power has increased, yet in the building industry we are still predominantly working in the same ways as we have always done. Instead of applying computational methods to existing workflow, how might computation help us to rethink the practice of collaborative design? That is the essence of this research.

1.1. EngD Beginnings

The first thing to point out to the reader is the inductive nature of this thesis. When the journey began, I did not begin with a problem question set by another person to form an hypothesis. I did not break this down into sub-problems and solve them individually, ending with a neatly packaged final result.

Instead, the initial research years allowed questions themselves to be emerge and develop in industry, arising during participation on many real projects that involved aspects of computational design. In this sense, the thesis aims and methodology initially adopted were intentionally loose, transforming to a more rigid structure as time progressed, much like the design process itself. The work undertaken was initially broad in scope, guided by the case study projects with questions and possible solutions becoming more specific as time progressed. The resulting outcome therefore bridges ideas from several disciplines to form a new approach presented towards the end of this thesis.

1.1.1. Ramboll and I

My sponsoring company during the EngD programme was Ramboll, an engineering consultancy founded in Copenhagen, Denmark in 1945. They have offices around the world, including the UK where the head office is based in London. At the time of writing, Ramboll has close to 10,000 employees worldwide, specialising in many disciplines of which buildings are a subset.

In 2009, Ramboll decided to fund research into computational design due to its growing importance in the wider community, resulting in this EngD. The decision to sponsor an EngD was due in part to an

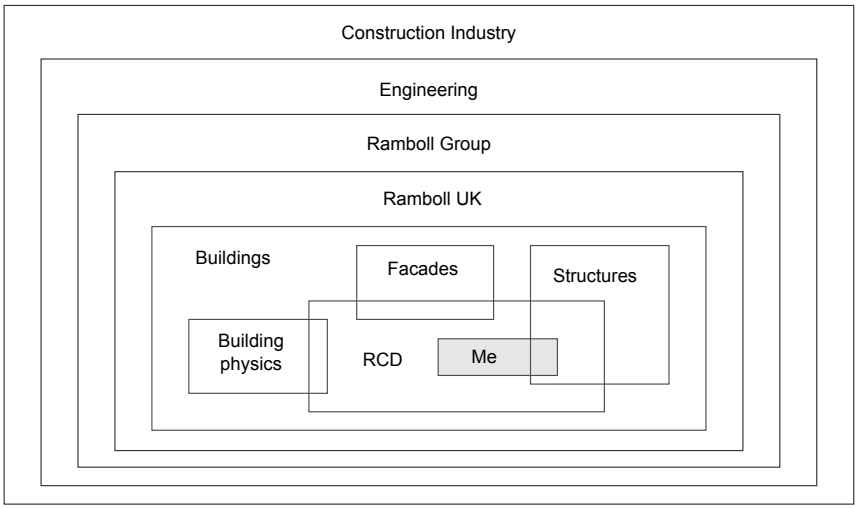


Figure 1.1: Location within industry hierarchy

increasing number of projects Ramboll were involved on with complex geometry, made possible by recent advances in computer modelling tools and fabrication methods.

During my research (and partly due to it), a small specialist team was founded in 2011 known as Ramboll Computational Design (RCD). At the time of writing, this team is still active and has gained wide acceptance within the company. Internally, the team works closely alongside the Structures, Building Physics and Façades teams at Ramboll (fig. 1.1) although the amount of involvement from each varies depending on the project. During my review period I actively attempted to become involved in projects that covered all of these disciplines either individually or through the course of several different projects.

1.1.2. Ramboll's Initial Intentions

However broad the subject matter, one must begin somewhere. Figure 1.2 shows the initial starting point for the research set out by Ramboll. The initial idea was to develop a generic multi-objective optimisation process in order to make all of our projects *better*. To be more specific, this meant making things perform better relative to measurable objectives, therefore helping to make our designs perform more efficiently both in terms of monetary costs (capital and operational) and environmental ones. The goal was to find the perfect solution to each given project; optimising until the computer reaches a single outcome - the perfect building.

Such a problem makes an assumption that we can write software that can optimise for anything, so long as we can put a number against it, know whether to minimise of maximise it and make each of these design objectives independent of one another. It soon became clear during my

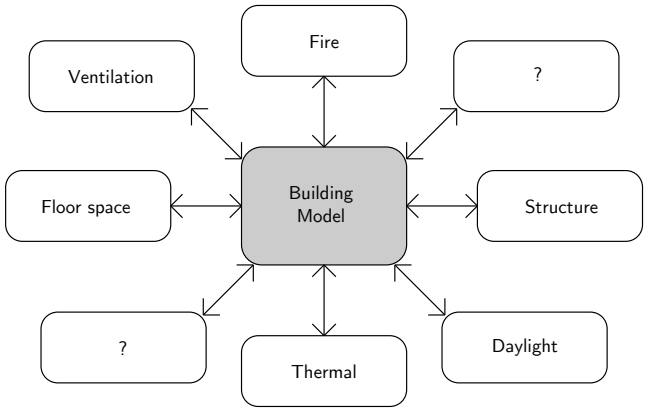


Figure 1.2: The performance of a building model measured by multiple quantitative objectives

literature review that this situation is far removed from reality, and the initial research diagram by Ramboll was somewhat idealistic. This led to a broader overall thesis objective as described in Section 1.2.

Even if we assume we can find an optimal solution to the problem of generating the perfect building, it may also seem tempting to think that the process used could potentially be generalised for all projects. In reality, there are always differences. Even if the final design is similar to previous work, the people involved in making it happen are likely to have been different. As engineers, the stage of involvement and how much subsequent freedom there is to influence design development becomes a dominating factor in setting up the problem, just as much as the measurable entities such as the site dimensions, the price of steel or the path of the sun. There exists a certain skill in understanding what human aspects could be generalised from project to project and which require bespoke treatment. In this regard, my initial review in Chapter 2 focusses on this generalisability by keeping a record of my primary experiences in practice combined with secondary research sources when attempting to approach computational design from an engineering perspective.

This inductive approach of letting the case studies guide the research allowed an initial freedom in understanding where the industry and myself were before making any purposeful interventions on future projects. My personal EngD story can be seen as similar to the design process itself; trying things out and allowing the direction to emerge over time.

1.1.3. What is Computational Design?

Computational Design can be described as the use of computing machines as part of the design process. This is not simply the use of

pre-compiled computer software such as CAD, but rather the creation and application of computer algorithms themselves which enable the vast numerical computing power of the machine to be utilised. In architecture, the field is also known as Computer Aided Architectural Design (CAAD), although Digital Design or Design Computing are also commonly used. It is a broad topic, influenced by design theory, computer science, mathematics, biology and engineering.

The term 'Parametric Design' is also sometimes used interchangeably with 'Computational Design' although with the former there is often more of a focus on the specific use of parametric modelling tools that employ visual programming techniques as opposed to direct coding to conduct design exploration. Indeed, there is still some misunderstanding with the terminology in the community, so for the purpose of this thesis I will state that parametric design is a subset of computational design that utilises an explicit form of representation. This includes dataflow programming, building relationships in a spreadsheet or a set of explicit instructions laid out in code, but excludes emergent processes inspired by biological systems (agent based models, cellular automata, etc.) with complex outcomes. A similar definition can also be found in the recent doctoral thesis of Davis (2013) who encountered a similar problem with the terminology. As we will see in Section 2.4.1, this thesis will eventually focus more heavily on the dataflow programming approach to parametric design.

Those involved in computational design need to be able to handle code (textually or visually), and know how to program computers to allow them to perform calculations. The experience and skills required to work with computational tools therefore may well be quite different to those needed for traditional design processes (Lawson, 2006). Whilst at its inception, human experimentation with computers used to be confined to labouriously programming university supercomputers with punched cards, advances in computer hardware have now made such machines available to all. This availability has resulted in the development of software applications to enhance our capability in existing design tasks such as drawing and visualisation, however little has changed in terms of how these machines influence design workflow in practice. Complex projects are often simplified through the sequential ordering of tasks - the extreme case being the architect developing the concept design in isolation, the engineer then attempting to *make it work* followed by the contractor and sub-contractors there to build it.

Computing however can offer new opportunities in questioning traditional thinking underpinning the design process in industry, allowing better collaboration from day one by incorporating the multiple needs of stakeholders (Shepherd *et al.*, 2011). This is also the central idea of

Building Information Modelling (BIM), however a BIM approach treats the computer more as a slave and not an active participant in design. One may contrast this perspective with Negroponte's *architectural machines* (1969) or Frazer's *electronic muse* (1995); treating the computer not merely as a clerk to human requirements and repetitive tasks, but rather a collaborator in the creative process with a child-like innocence.

As computer algorithms calculate incrementally, defining a *process* becomes just as important as the final outcome. This focus on process means computational design is frequently compared to the development of natural organisms. Indeed, this link to biology has led to terms such as *Morphogenetic Design* (Hensel *et al.*, 2004) or more recently and almost identically *Digital Morphogenesis* by Leach, who comments that architectural discourse in general is moving to "engage more with science, technology and material behaviour" (2009, p.37). This is especially the case in light of the current drive towards environmentally responsible design and how computers can collaborate as part of the design process. Here the engineer can begin to offer computational methods at the front end of projects that go beyond the architect's own knowledge, however exactly how this is implemented in collaborative practice is a complex task.

1.2. Thesis Objective

1.2.1. Computational Design

How can computational methods assist the design team within collaborative practice?

As described in Section 1.1.2, although the initial thesis objective set out by Ramboll was in multi-objective optimisation, it soon became clear throughout year one that if the thesis was to be driven by the projects then the problem statement would have to become more general. This was because the people involved on the projects with their qualitative judgements and intangible motivations were not a good fit for the creation of numerically optimised perfect buildings. Although the resulting statement therefore became quite broad, it was still important to form at least some initial boundary judgement (Churchman, 1970) to set out what was to be included and excluded in my research.

Figure 1.3 shows where the initial problem question sits within the wider context of the projects, and how this question is then broken down into two sub-questions relating to hard and soft systems. The initial aim was to investigate the current state of technology and its application in

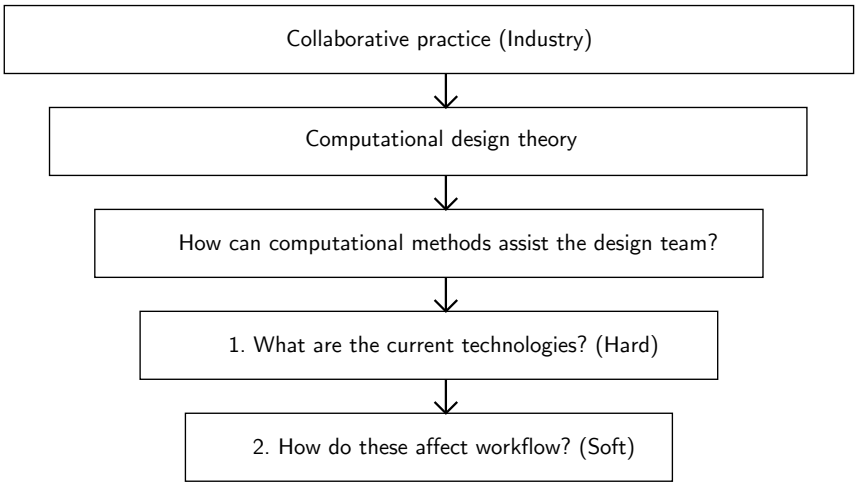


Figure 1.3: Question hierarchy within wider context

practice. It is clear that advances in computational design will affect the project workflow, however the projects themselves will also influence the technology that has been developed.

1.2.2. Meta-Parametric Design

The first part of this thesis (Chapters 2-5) records the development of a more specific research topic through a series of case study projects. This later work focusses on the *Meta-Parametric* approach and an associated software application named *Embryo*, developed by the author and discussed at length in Chapters 6 & 7. The idea concerns the automatic generation of parametric model definitions themselves, thus moving to a higher level of abstraction to broaden their scope.

The thesis question to be addressed in Chapters 6 & 7 is the following:

> *How does the use of a Meta-Parametric approach assist design teams at the concept design stage?*

The parametric design software Grasshopper for Rhinoceros (Robert McNeel & Associates) is adapted using Embryo in order to address this question. We will see that enabling the automatic generation of parametric model definitions leads to interesting questions of control, ownership and understanding of the model. In the case study examples given in Sections 7.1 and 7.2, a series of machine generated designs are compared on real projects at the concept design stage. The suitability of these generated designs is assessed in terms of the following aspects:

1. The design space of a generated model (variability).
2. The best fit to the objectives (performance).

3. The complexity of the graph (intelligibilty).

We will see that simple parametric definitons with clear causality are preferred by design teams so that control can be regained over the generated models by understanding them. This desire is balanced against generating suitably complex models for the design problem. Essentially, Occam's razor seems to apply to the parametric definition even if the generated geometry itself is complicated, through choice and/or necessity. This limits the scope of a Meta-Parametric approach using software such as Grasshopper if one is to regain full control; however if 'let loose' so to speak, automatically generating models can still have benefits as pure design explorers.

1.3. Methodology

Addressing the initial thesis objective sub-question 1 (fig. 1.3) requires a traditional approach to the literature review, citing secondary sources in order to see what has come before. These technologies are then applied in practice in order to work on sub-question 2.

Sub-question 2 requires more consideration as to the application of technology in its context, that is, not just the computational approach but the consequences for the stakeholders involved as part of *real-world problems*. Adequate understanding of the design process cannot take place using reductionist methods alone. Instead, the use of inductive reasoning in order to arrive at some broader generalisations will then enable suitable interventions and response. This will be combined with examples from industry in order to construct a sound base with which to acknowledge design as an *interpretive system*. An interpretive systems approach respects the world view of multiple stakeholders which may well be pluralist, that is, participants do not necessarily share the same values and beliefs (Jackson, 2000). Acknowledging multiple stakeholder world views is crucial for understanding the problems and necessary response when using computational design techniques in a collaborative environment.

1.3.1. Data Triangulation

The concept of triangulation is that general conclusions made throughout this thesis should come from three of more sources. Combining both primary and secondary data sources is a good way of ensuring that findings may well be general, as well as helping to remove researcher bias

(Robson, 2002). As a general rule, within this thesis I have attempted to combine one or two case studies with one or two findings from literature of similar problems in industry in order to triangulate my results.

1.3.2. Primary Research Methods

Case study projects form the backbone of the initial primary research into the current state of computational design in practice. There are 33 projects in all for which I was an active participant, most of which took place in the first two years of the research. Each one involved different stakeholders, some internal to Ramboll and some outside the company such as architects and clients. At the end of the initial review period an active effort was made to intervene at the earlier stages of projects (fig. 1.4).

The fact that I was involved multiple case studies and at different stages is important. Multiple projects help to reveal general trends that are common in the practice of computational design on live projects, not just theoretical examples. This approach meant it was the case studies that indicated recurring aspects of computational design methods, eventually leading to the distinctions laid out in Chapter 2. Instead of hypothesis testing as per the scientific method, the research direction emerges inductively from a practice-based context (Fook, 2002). To this end, the details of each project were recorded throughout, with a summary for each project found in Appendix A. The information included for each is the following:

- General project information (i.e. project sector, size, value, year of my involvement, etc...)
- Stakeholders involved on the project, both internal and external to Ramboll
- Any computational design approaches used.
- Relevant software/ programming language used.
- Short description of my involvement on the project

During the case study projects there were two main methods of understanding the human aspects when adopting computational design methods: *participant observation* and *semi-structured interviews* (Robson, 2002). The former was conducted during project work and the latter towards the end, and included stakeholders both internal and external to Ramboll. Due to the nature of the projects, it became necessary to declare my role as part of the EngD, and hence not become what is known

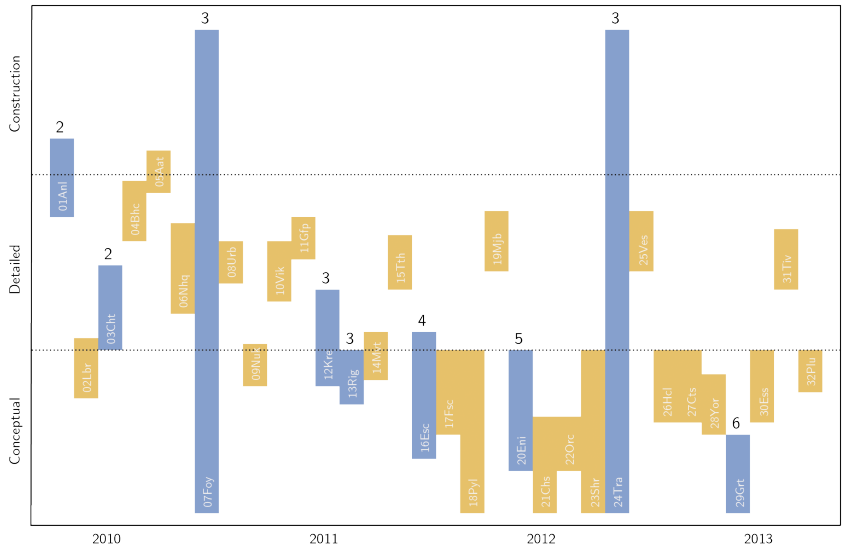


Figure 1.4: Case Study Projects: initial involvement date and design stages shown. Projects shown in blue are major case studies with the relevant thesis chapter indicated

as a *complete* participant (Robson, 2002). It made no sense to conceal my status as a research student because I was treated as part of the Ramboll Computational Design team anyway by having worked for the company before taking on the EngD. In addition, I quickly established there was nothing to be gained from coercive behaviour because it is clear that there would be a benefit to all parties should the research be successful. As described in Section 1.1.1, my role as a participant was from within the Ramboll Computational Design team. To better understand the viewpoint of other stakeholders on projects, participant observation alone would not have given a true understanding of the whole picture, and so it was felt additional interviews were required.

In order to get the best results, semi-structured interviews were conducted with design team members in a setting familiar to them (Robson, 2002). The use of semi-structured as opposed to structured was to give the interviewee freedom to take the direction of the conversation to places that was unexpected. This approach was particularly successful on the ENI project [20Eni] described in Chapter 5. First hand observations were recorded using personal reflective logs, blog posts and by storing all material from each project even if design directions were abandoned.

The primary methods used assisted my understanding beyond secondary research because participant observations and interviews with stakeholders during actual projects can provide details that are not often published through written material such as project write-ups or academic papers. There may also be issues of confidentiality or issues with divulging information on how particular projects are designed for general publication. The real-time observation therefore helped to be able to record real emotion and subsequently proved to give important findings later in the research. By becoming a participant myself inside the

system and not an outside observer, it was possible to better understand what was really going on throughout projects and not just what people were willing to tell me.

1.3.3. Secondary Research Methods

In addition to the case studies, peer reviewed published material forms an important part of the research into the current state of practice. Such research generally falls into the following two categories:

1. Research relating to a computational design approach (hard/structured)
2. Research relating to the application of a computational design approach (soft/interpretive)

Clearly these two categories overlap. However, as discussed in Section 1.2, secondary research material relating to the hard aspects can often exist without reference to its application in industry. This work tends to be conducted in academia alone and formed a vital part of my own learning and experience from a technical point of view. In contrast, the softer *social* aspects of applying computational design techniques cannot exist without reference to the particular *hard* technology that is used. Secondary research sources commenting on application often cite project case studies in order to illustrate why a particular approach has been successful or not, describing in detail the limitations of the particular software/approach used. For these *soft* aspects, gathering data from industry about real-life computational design helps both remove bias from participatory research and understand whether a project specific method has a more general application elsewhere.

1.3.4. An Adaptive Methodology

This use of inductive reasoning is continual, with technologies, people and their environment in a continual state of flux. Due to the nature of the EngD, with such a mixture of hard and soft systems and no one clear objective from the outset, it is somewhat inevitable that the research will take unexpected turns along the journey. As Gill and Johnson state (2002, p.9):

> "The research process is not a clear cut sequence following a neat pattern, but a messy interaction between the conceptual and empirical world - deduction and induction occurring at the same time."

The methodology attempts to look at the subject matter holistically and always within a wider context. This isn't without risk, and certain parts of this research could no doubt have been investigated to a finer detail. As Erwin Schrödinger remarks at the beginning of in his anti-reductionist work 'What is life' (1992, p.1):

> "...some of us should venture to embark on a synthesis of facts and theories, albeit with second-hand and incomplete knowledge of some of them - and at the risk of making fools of ourselves"

This is not an excuse for not adopting a strict approach, but rather an early acknowledgement that when thinking laterally across disciplines, knowing everything about everything is not a realistic proposition. In view of the nature of an EngD, I therefore review the aims and objectives at the end of each chapter as my research progresses. These changes inevitably lead to supplementary literature reviews being undertaken throughout the thesis. I have however attempted to retain a chronological structure, accounting the progression of thoughts and findings throughout the research journey.

1.4. Contribution to Knowledge

This work aims to contribute to the field of computational design in general, but also due to the nature of the engineering doctorate, to my sponsoring company Ramboll. As an EngD, an emphasis is placed on the application of technology as well as the technology developed itself. The key contribution areas are the following:

- Advancement of knowledge within Ramboll. Knowledge capture by building a body of work for the company to be able to better understand the application of computational methods in practice and how to work with other consultants. This includes computer programs and the learning from their application. Knowledge shared more widely at Ramboll by presenting internally to the company at the annual research conference.

- Advancement of knowledge to the wider construction industry. The learning of other key stakeholders during the projects. Presentation and publication of findings within the industry. The general release of computational software tools developed to other companies and research institutions.

- Advancement of knowledge in academia. Papers and conference presentations to the computational design community to disseminate any relevant findings and approaches that might have wider benefit.

1.5. Chapter Outline

This thesis should be read as a continual narrative with changes in direction along the way. A brief summary of the chapters is as follows:

Chapter 2 is a review of the current situation for engineers involved in the field of computational design, both in terms of technology and its application in practice. It looks at both direct methods which embed engineering performance inside the algorithm in order to optimise performance, and also methods which externalise objectives, using meta-heuristic search algorithms. I conclude that in their current state, whilst the methods developed are sophisticated, they do little to challenge the linear design process already established between concept design and problem solving (making the concepts work). This finding encourages action to place myself further towards the front end of projects.

Chapter 3 involves purposefully working earlier in the design process as an engineer, first by attempting to go alone in the development of computational design approaches for projects and then by developing software for other consultants which embeds certain engineering constraints and hence directs the design of form. Whilst this is successful, it is found that such approaches are suitable to embedding only a single aspect of building performance, and hence not allowing for the complexity of architectural design. As a result, the next chapter investigates more general modelling methods such as those used in parametric design.

Chapter 4 describes the experience of using of generic software such as parametric models that can allow multiple stakeholder involvement when investigating building concepts. It is found that although they have a useful explicit representation, they are often inappropriate for the concept stage due to their inherent topological inflexibility.

Chapter 5 describes the development of a complex modelling approach that allows for flexibility in building typology in contrast to parametric models. This is tested on a particular case study project. The approach was not successful and the decision to reject using parametric models outright was reconsidered.

Chapter 6 is the final proposed response, a dialectic meeting of Chapters 4 and 5, with the development of the 'Meta-Parametric' approach, suited to the early stage of design. This method involves the automatic generation of the directed acyclic graphs used by popular parametric modelling software (such a Rhino Grasshopper). A plug-in for Rhino Grasshopper called 'Embryo' is introduced.

Chapter 7 describes the testing of Embryo in three situations. The first two involve real projects at the concept design stage, conducting a wide

exploration of form with unknown goals. The third is using Embryo on a shape analysis problem for a clearly defined goal, opening up the possibility of generating alternative parametric definitions for existing models.

Chapter 8 will discuss the findings of the research in relation to the meta-parametric approach. Some suggestions for further work are proposed.

Part II.

Review

2. Post-Rationalisation

> "I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail"
>
> Abraham Maslow (date unknown)

2.1. Introduction

This chapter covers an initial review of computational design with respect to an engineering practice. This includes most of the case study projects conducted at Ramboll within the first year of the doctorate, but also a review of literature on both technical aspects (hard) and collaborative aspects (soft) that are associated with computational design. The initial set of case studies involved coming on board late in the design process, taking existing architectural conceptual designs and improving them in terms of environmental and structural efficiency. To begin, Section 2.2 discusses case studies involving the engineering optimisation of a single objective using a heuristic embedded in the computational approach.

Section 2.3 describes several case studies where the engineering goals are not embedded within an algorithm. Instead, a problem is formulated and metaheuristic algorithms are used in order to search for solutions. The formulation of the problem constraints and objectives become crucial to finding an appropriate solution.

Section 2.4 discusses the use of parametric models in the context of metaheuristic algorithms, whereby parameters can be adjusted manually or automatically according to performance criteria. In Section 2.5 this is extended by considering optimisation problems with multiple objectives and their use within a collaborative environment. For such problems, multiple performance criteria are often non-linear, non-mutually exclusive, and driven by different stakeholders on the project.

2.1.1. Building Models

In comparison to industries such as software engineering where the final outcome can be modified and future releases made, getting building design right is usually a one-time operation. Any subsequent changes due to mistakes can become costly following handover. Likewise, one cannot easily build a full-scale building prototype for testing, especially if the project is complex and unique to its context. Getting things right before commencing construction is therefore favourable and abstract building models that *simulate* behaviour (aesthetic, structural, operational, etc...) are used. Architects and engineers typically rely on computer building models in order to iterate through the design process and finalise a set of drawings that are tendered for construction.

In recent years, advances in computer software have enabled architects to investigate geometric forms that are only limited by their imagination. This surge in form-making has led to interesting and challenging problems that engineers are there to solve. The following case studies highlight this to be the case in industry. Building models are often received from the architect following the completion of the concept design stage, either in the form of raw geometry files or parametric models. Upon receiving a model from the architect, the task of the engineer is often to *make the concepts work*, with various computational processes deployed in order to assess and if possible improve the engineering aspects of the design. Such involvement can be classed as 'post-rationalisation', in that the design concept is already fixed. Building models received from the architect already provide the problem constraints within which an engineering objective is pursued. The use of computation to solve such post-rationalisation problems in practice typically falls into two categories: heuristic and metaheuristic algorithms which are discussed in the following sections.

2.2. Heuristic Algorithms

2.2.1. Introduction

In computer science, heuristic methods are techniques for solving problems that include some sort of rules of thumb or experience embedded in the algorithm. Such methods often attempt to optimise for a single goal. They do not always guarantee a completely optimal result because complete innocence of the engineer is not assumed. Instead, some experience as to what direction to take is embedded into the computational process itself in order to get results in a reasonable time. Based on the case study

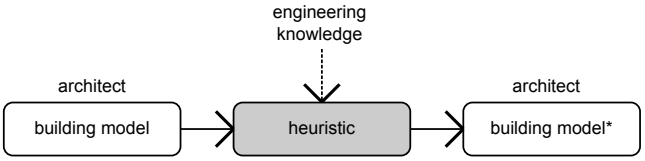


Figure 2.1: An architect's model is modified using an algorithm with embedded engineering knowledge. The grey shade indicates engineer influence in the process.

projects undertaken, the use of heuristic algorithms has been divided into the following 3 categories:

1. Heuristic without external analysis: an algorithm that arrives at a result independently of outside information.

2. Heuristic with consistent external analysis: an algorithm that uses outside information that remains constant.

3. Iterative external analysis: an algorithm that uses changing external information updated at each iteration.

2.2.2. Without External Analysis

Perhaps the simplest use of a computational method is where the engineer deploys an algorithm in order to improve a current building model (fig. 2.1). The algorithm used may embed some specialist knowledge or experience in order to produce a desired outcome that is more or less known and expected. The use of a computational approach is beneficial in that outcomes can be reached that go beyond manual methods, due to the iterative power of the computer. Structural form-finding algorithms are a common example of heuristic methods that do not use any external analysis, and were used in multiple case studies during my review period. These algorithms tend to address problems that cannot be solved analytically and require a numerical approach.

Form-finding Minimal Surfaces

Structural form-finding defines techniques for developing geometric form by embedding structural heuristics. As opposed to manipulations made externally by the design team to improve structural performance, structural logic is instead embedded in the form-generation process itself. One of the most famous examples in architecture is the problem of finding a minimal surface within a set of boundary conditions. It is known that for lightweight fabric or cable-net structures, minimal surfaces give a constant tension form when pre-stressed (Lewis, 2003); however, finding minimal surfaces for non-trivial boundary conditions is not a simple problem and requires a numerical approach. The structural heuristic that is used in this case relies on the relationship between

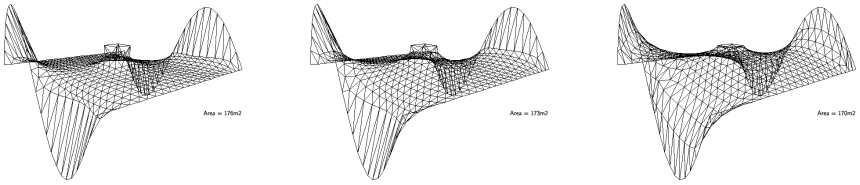


Figure 2.2: Finding a minimal surface for the Garden Festival Pavilion

minimal surface geometry and the associated structural benefits for lightweight structures.

The 1972 Munich Olympic Park tensile roofs by Frei Otto and Günther Behnisch are perhaps the most famous examples of lightweight structures with form-found minimal surfaces (Otto & Rasch, 1995). Although such forms were originally found using *natural computing* methods such as physical models (in this case soap films), numerical methods using computers are now favoured due to practical considerations such as the ease of boundary condition manipulation and the input/output of data (i.e. node positions).

For the Garden Festival Pavilion [11Gfp], an algorithm was used to find a minimal surface from an initial *best guess* surface geometry. In the approach, discrete triangular elements can approximate a continuous surface using the *triple-force method* (Barnes, 1999; Lewis, 2003) in combination with dynamic relaxation (Day, 1965). The equilibrium solution is found through iteratively applying residual forces until the system converges (fig. 2.2). For this particular project, a Java application was written by the author that imports an initial mesh, finds the minimal surface and then exports the result. The final geometry was then issued to the architect.

Laplacian smoothing, a simpler form-finding process was required for the Lusail Bridge Roof project [02Lbr]. This is a well-known process in computer graphics that smooths an initial polygon mesh by averaging the position of neighbouring vertices. Rather than solving for engineering benefit in this example, the process was to generate a desired anticlastic shape (i.e. with negative Gaussian curvature at all points) as requested by the architect, in this case our own bridges team within Ramboll. The final result is shown in fig. 2.3, with the final BSpline surface form used for further design development. Laplacian smoothing was also used on the Urban Bubble Gridshell project [08Urb] in order to produce a smooth mesh for the underside of an ellipsoidal form which had to met the ground at 4 locations. In these two case studies, aesthetics was the main driver behind the mesh smoothing, although they also had the added structural benefit of reducing discontinuities.

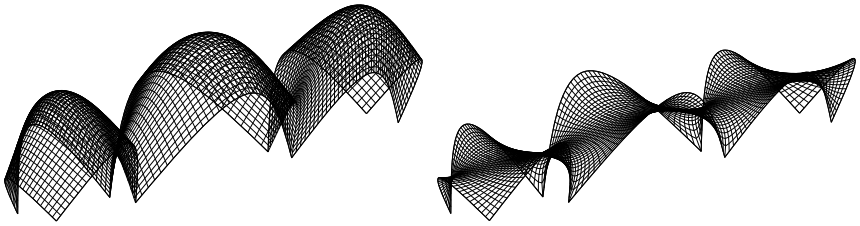


Figure 2.3: Laplacian smoothing on the Lusail Bridge Roof

Funicular Form-finding

It is well known to engineers that for structures whose dominant load case is self-weight, a system that works by transferring load using axial forces is more attractive to those that work in bending in terms of minimising material use. The method of finding such forms can be traced back to Robert Hooke who once famously wrote (as an anagram) "As hangs the flexible line, so but inverted will stand the rigid arch" (1675). Antoní Gaudi's physical hanging chain models (Collins, 1963) have subsequently inspired many architects and engineers to use similar methods in funicular form-finding including Heinz Isler (Chilton, 2000) and Frei Otto (1995). A more detailed literature review of funicular form-finding approaches can be found later in Section 3.2.1, where the subject is addressed in more detail as a main driver behind the design process as opposed to improving existing designs.

Directly replicating the behaviour of physical models in the computer was applied during various case study projects using bespoke Java applications written by the author. This occurred both in two dimensions, for example on the York Way Project [28Yor], and in three dimensions for both the Markham Vale Sculpture [14Mvt] and The KREOD Pavilion [12Kre] (fig. 2.4). For these particular case studies, an *existing* geometric form was post-rationalised to improve its structural performance. An initial structure supplied by the architect is freed at the nodes (creating fully pinned connections), and subjected to an inverse gravitational load. The resulting large displacements were solved using a relaxation algorithm written by the author (See F.1) that treats bar elements as stiff rigid springs, based on a similar real-time method by Kilian and Ochsendorf (2005). When the gravitational field is reversed, the structure works more efficiently under self-weight than before by increasing axial force and reducing bending moment. For these three case study projects, the resulting geometry was then sent back to the architect for approval.

Form-finding for Fabrication

As well as finding structurally efficient forms, several of the case study projects involved discretising surface based geometries for fabrication

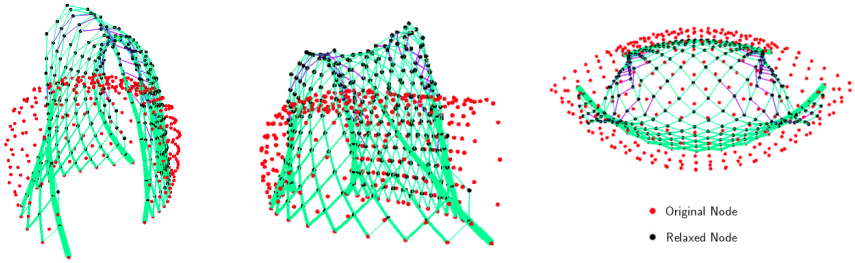


Figure 2.4: Initial surface form is relaxed on the KREOD pavilion to improve structural behaviour of the final design

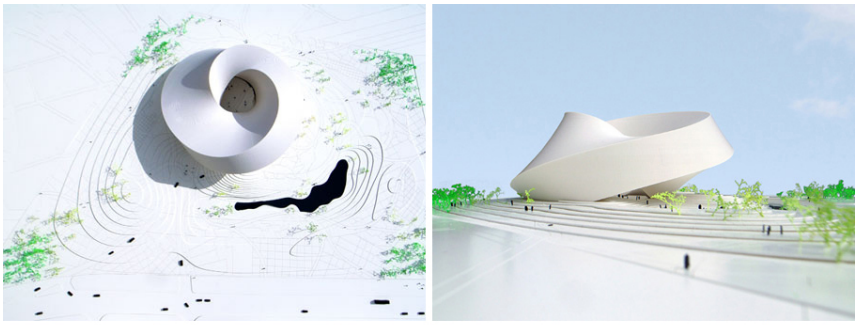


Figure 2.5: Architectural concept for The Astana National Library

and construction. The heuristics used were again embedded in the algorithm itself. For example, the distribution of nodes, bars and surface elements to discretise a surface is a common problem in façade design. In this section, form-finding on the projects occurs within a surface embedding space.

On the Astana National Library, [01Anl] my involvement began after the architect had already developed a continually twisting envelope geometry by joining two Möbius strips along their boundaries (fig. 2.5). This created surface geometry that although was ruled (and therefore advantageous for the supporting steelwork), meant the façade surface itself was doubly curved and therefore undevelopable, i.e. it could not be unfolded to the plane without stretching or tearing Flöry & Pottmann (2010). A triangular discretisation was therefore adopted, formed by using the surface ruled lines. This approach however led to every façade panel being a different size and shape (fig. 2.6).

In response, a short discretisation study was undertaken that used a *bottom-up* repulsion based meshing algorithm without a fixed topology. This approach is described by Turk (1992) in the generation of fair meshes, and was later applied by Kanellos (2007) and Lewis (2011) in generating three-dimensional structures. A particle repulsion method with a fixed topology was used by Williams (2001) when relaxing the triangulated mesh on the British Museum Great Court Roof.

Structural nodes are initially located randomly on the surface and are then repelled from nearest neighbours iteratively. The strength of repulsion is gradually damped until equilibrium is reached with an

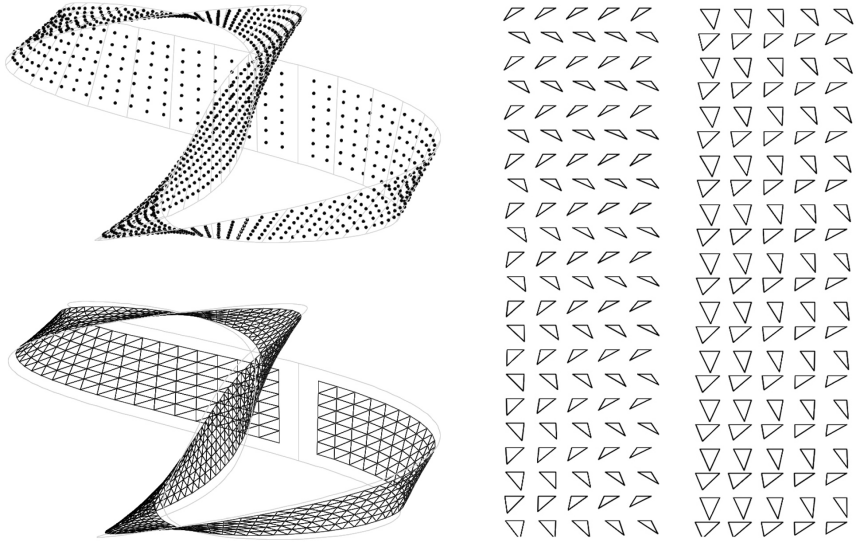


Figure 2.6: Initial Astana National Library façade tiling. Although the base surface is rational, the proposed panels were all a different size and shape changing gradually as they move around the surface.

equal force of repulsion between neighbouring nodes and therefore equal distance. The approach is similar to the annealing of metals, whereby the rate of cooling creates particular atomic spacings and therefore different material properties. The manually defined damping rate must be above a critical value, however this can usually be obtained through visual inspection. This repulsion process is a form of self-organisation, because global order arises out of the local interactions between the nodes which are initially disordered. Figure 2.7 shows one section of the library façade surface, indicating the stages of development with edges connecting nodes neighbouring nodes that are currently influencing each other.

A final triangulated topology is then formed by finding the six nearest neighbours for each node, and removing any edges above a certain cut-off length. Due to the equal distance between nodes, this triangulation has a high proportion of same sized edge lengths and equilateral triangles, leading to manufacturing cost savings and reduced complexity during construction. The amount of identical edge lengths for the Astana National Library façade surface shown came to approximately two-thirds of all 600 edges.

In such examples, an optimal solution may exist but in practice, simply being able to achieve a *better* result than a more direct approach (such as triangulating using an isocurve grid), can have a large impact. For Astana, the knowledge of what the final geometry *meant* to the design team was embedded in the algorithmic approach.

Even though improvements can be made, the nature of the shape meant the façade complexity remained relatively high. Essentially, the underlying concept geometry of the twisting form generated issues further down the design process that were not initially apparent.

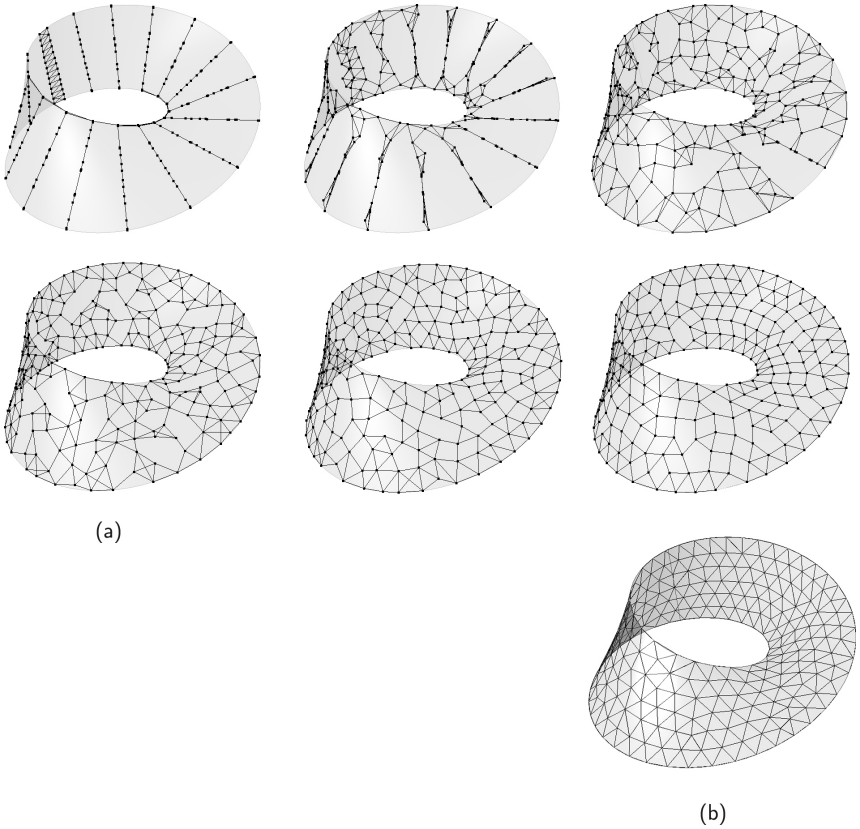


Figure 2.7: Nodal repulsion process on Astana National Library (a) and the final triangulated mesh (b)

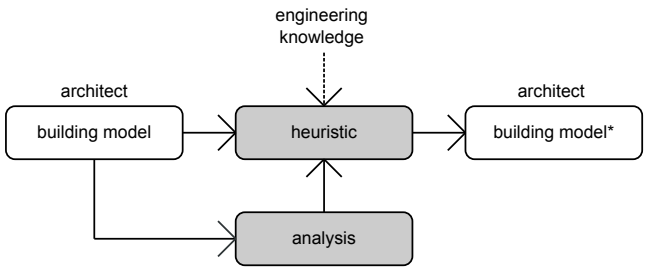


Figure 2.8: Algorithm that incorporates external analysis information

2.2.3. Consistent External Analysis

In the previous section, all of the engineering knowledge behind the approach was embedded in the algorithm itself. However, other methods enable external information such as structural analysis to take place during computation. In this section, some type of external information is required beyond what can be provided by the engineer within the algorithm, with the condition that this *external analysis* information stays consistent and does not change during an iterative process (fig. 2.8).

Façade Design

Curvature analysis of an architectural surface form can lead to fabrication and structural benefits when discretising into smaller elements. For example, aligning quadrilateral panel edges to the principal curvature

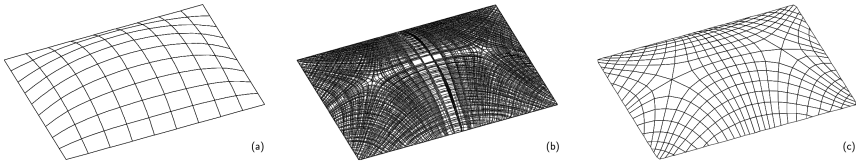


Figure 2.9: A freeform doubly curved surface (a) with principal curvature lines found (b) gives PQ mesh of appropriate density (c)

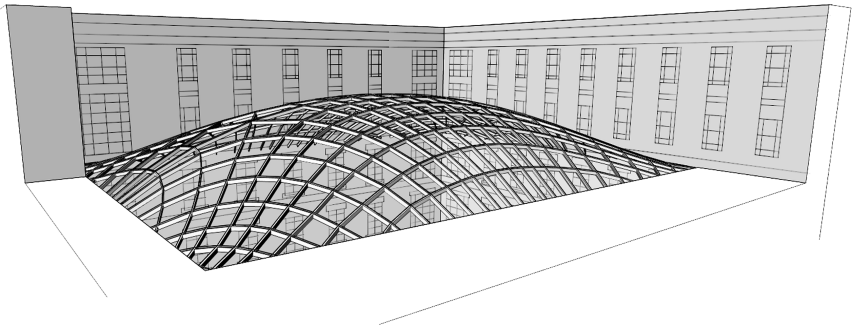


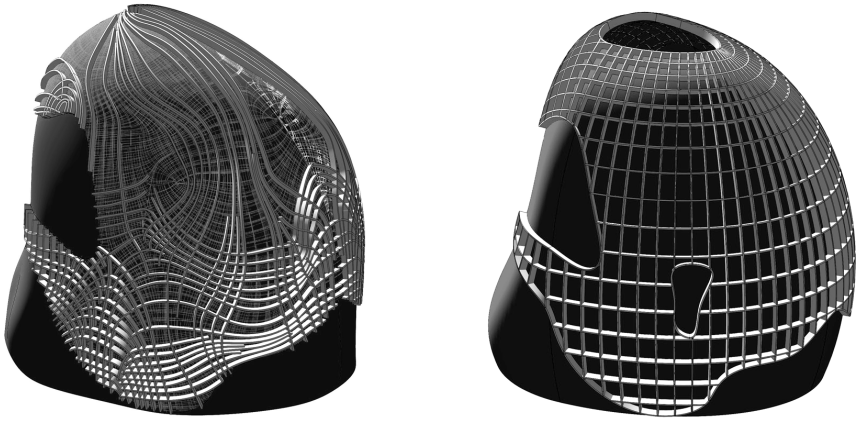
Figure 2.10: Final design for the Senate House glazed roof made from planar quadrilaterals

directions (describing minimum and maximum curvature) leads to them being planar, generating a Planar Quad (PQ) mesh. This planarity and lack of triangular panels give cost benefit in terms of manufacture, especially for glass. In addition to panel planarity, supporting members are torsion free (i.e. no twist along its length) with the additional benefit of members meeting at a common normal, thus improving the connection detail both in terms of cost and complexity (Liu *et al.*, 2006).

The principal curvature field for a freeform surface must be found numerically, and hence an iterative approach is required in order to generate the streamlines of the curvature field that generate the pattern. A generic algorithm that generates the conjugate curve network using surface curvature analysis was written by the author to assist on various case study projects (fig. 2.9).

This approach was used and refined during various projects including National Holdings HQ [06Nhq], London Foyer Sculpture [07Foy], initial work on the KREOD pavilion [12Kre], Tivoli Edge Development [31Tiv] and Senate House Roof [23Shr] (See fig. 2.10). The projects enabled close collaboration between myself and the Ramboll UK Façades team during the initial research period. The cost benefits of planar quadrilateral glass panels as opposed to hot-bent glass is significant. As Mark Pniewski, head of Ramboll UK Façades comments, "the cost difference between flat and doubly curved panels is currently around 8-fold for the same given area" (pers. comm., March 2014).

The Orchid House project [22Orc] gave a slightly different set of circumstances. For a doubly curved shape, a torsion-free member layout was required for the timber frame, however the principal curvature method produced an overly complex pattern (See fig. 2.11). By going beyond our remit as engineers, I found that some small changes to



(a) (b)

Figure 2.11: Orchid House [22Orc]. Initial torsion-free member layout (a) is rationalised through small modifications to the underlying form (b)

the surface geometry produced vastly different outcomes to the final principal curvature field and hence the associated PQ mesh. These modifications revealed a more rational member layout was within reach without large changes to the form. Returning to the architect, we were able to convince them that these modifications to the form should be made.

Here was an example of the building form itself being allowed to be manipulated by the engineer as part of a design exploration process, with the fabrication and structural implications revealed using a computational approach. This finding would go on to shape much of the work in Chapter 3, in particular the KREOD pavilion [12Kre] whereby the engineer attempts to get involved earlier in the design process *before* surface geometries are fixed.

Structural Design

As well as geometric methods which indirectly pursue a particular heuristic, a computational approach may include integrating an engineering property directly in order to propose an efficient solution. One such example is with the efficient placement of material along the principal stress trajectories of a continuum. This is known to occur in natural systems, such as the distribution of bone material (Meyer, 1867). A more detailed review of the subject is given in Appendix B.

Michell (1904) was the first to investigate using principal stress grids for generating optimal cantilevers. Later work by the engineer/architect Pier Luigi Nervi (1965) took inspiration from such natural processes and explored the idea of minimising material usage by aligning concrete ribs along lines of force, notably the application for the Gatti Wool Factory along lines of principal bending stress. As well as flat slabs, more

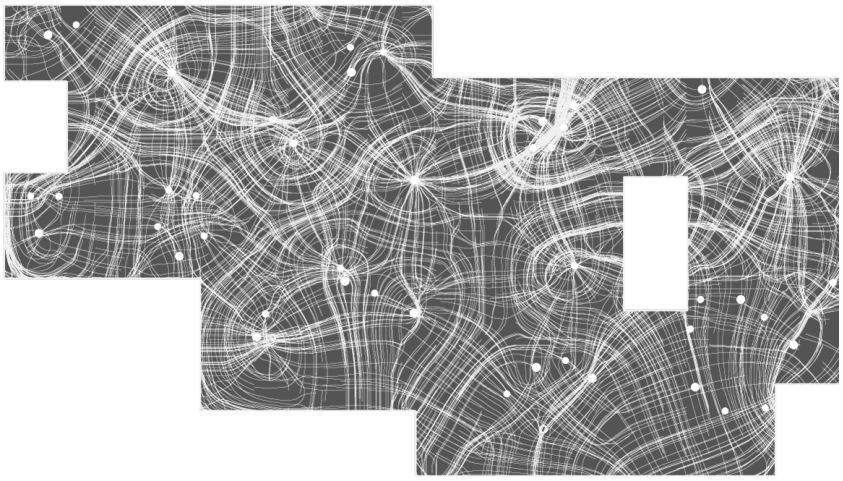


Figure 2.12: Helsinki Central Library competition: conjugate curves following the principal bending stress field

recent work by Kaijima & Michalatos (2007) and Winslow (2009) has investigated discretising freeform surfaces with principal stress grids.

I utilised this structural heuristic on several case study projects. These included The Greenland National Gallery of Art [09Nuk], the Viiki Synergy Building [10Vik] and The Helsinki Central Library competition [26Hcl]. Such structural drivers behind design often gave aesthetically interesting results (fig. 2.12), as well as encouraging the use of exposed concrete soffits often used for passive ventilation in buildings.

Passive Solar Design

As well as structural concerns, some projects during my review investigated using passive solar design principles to guide a computational approach. Algorithmic methods for generating architectural surfaces from solar paths have been used by Shepherd (2009) and Turrin et. al. (2010), although there are countless more examples.

Working with the building physics team at Ramboll UK, the Maljevik Bay Resort project [19Mjb] involved generating 30 roof designs for otherwise identical villas that were orientated differently on a hillside. Knowledge of passive solar design was incorporated in sculpting the roof forms for the architect according to the path of the sun both in winter and summer. In winter, sunlight into the building was maximised in order to increase the amount of heat and light generated through natural means. In summer the opposite is desirable. Solar gain and solar glare should be avoided at peak times during the day, hence the roof form should shade the main façade.

Figure 2.13 shows software developed by the author that *carved* an existing roof profile supplied by the architect. The algorithm can be

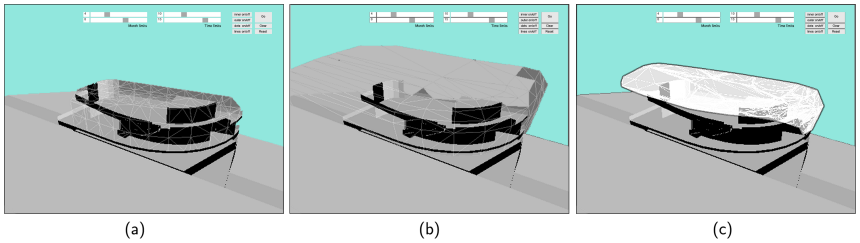


Figure 2.13: Java application for generating passive solar roof forms. Ray lines from a glazed façade (a) intersect with an extended roof surface (b). The resulting point cloud forms the new roof surface (c)

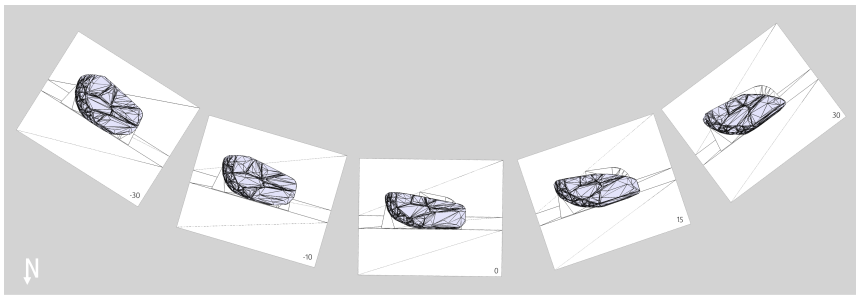


Figure 2.14: Maljevik Bay Villas: roof shapes generated for different orientation angles.

found in Appendix F. The generated roof form could then be exported a CAD file for each unique roof profile. The algorithm works by intersecting sun paths with an initial extended surface mesh (provided by the architect), generating a point each time. Once all the specified months, days and times have been calculated, the resulting point cloud is converted to a revised mesh surface using a constrained 3d convex hull (the constraint avoided large thin triangles). Figure 2.14 shows various roof forms generated at various different orientations on the site.

2.2.4. Incremental External Analysis

In the previous section, all the algorithms developed used some sort of external analysis, however this information was not dependent on the changing state of the model after each iteration. There was no coupling between the model and its environment. In this section, several projects are described where the geometry model is linked directly to any particular analysis so that any change to the model will affect the analysis results (fig. 2.15). Likewise, these analysis results in turn affect changes to the model. In terms of external information to the algorithm and its environment, there exists a *feedback loop* in the process. This feedback often gives greater uncertainty in terms of where the process is going and hence the final results are often unexpected.

Structural Geometry Optimisation

One process utilised during my research was that of node pushing to high stress areas, a heuristic method to improve structural performance of an

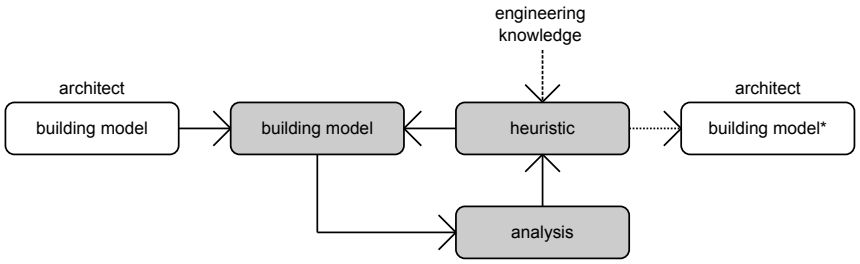


Figure 2.15: Algorithm using external analysis updated at each iteration. A cycle between modelling and analysis is now present.

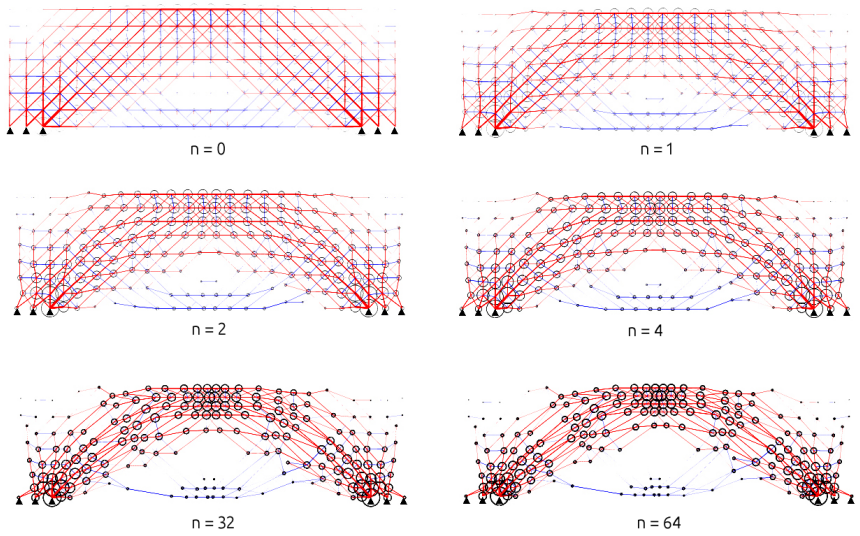


Figure 2.16: Node pushing for a simply supported beam results in a tied arch

existing node-bar configuration by altering its geometry (not topology). The method I used was similar that described by Kajima and Michalatos for the Land Securities Bridge project (2007), although instead of a constant stress field the structure is analysed at each time step.

The approach works by pushing nodes towards topological neighbours that are the most stressed at that iteration until a cut-off distance is reached (to prevent the merging of nodes). This cut-off distance specifies the maximum density of nodes in the structure. The algorithm written by the author can be found in Appendix F).

An initial test example is shown in fig. 2.16. A simply supported truss under self-weight morphs into a tied arch - a known efficient typology this problem. Rather than suggesting a tied-arch using engineering experience, it is interesting that a bottom-up approach results in an identical solution. As the simple test arrived at a *good* result that is validated by engineering experience it suggests that applying the same general heuristic might have benefits for less trivial problems.

In addition to the nodal repulsion approach (see Section 2.2.2), nodal pushing to high stress areas was also studied for the Astana National Library A structural façade. A Java application was developed that imported the library's façade geometry from Rhino, ran the node push

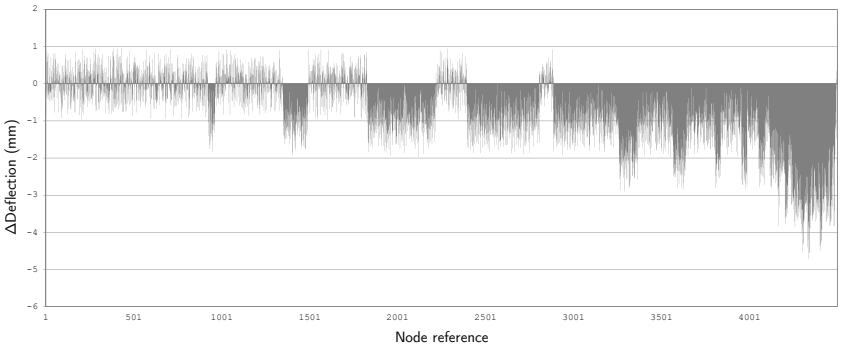


Figure 2.17: Nodal displacements difference between original and optimised on Astana National Library.

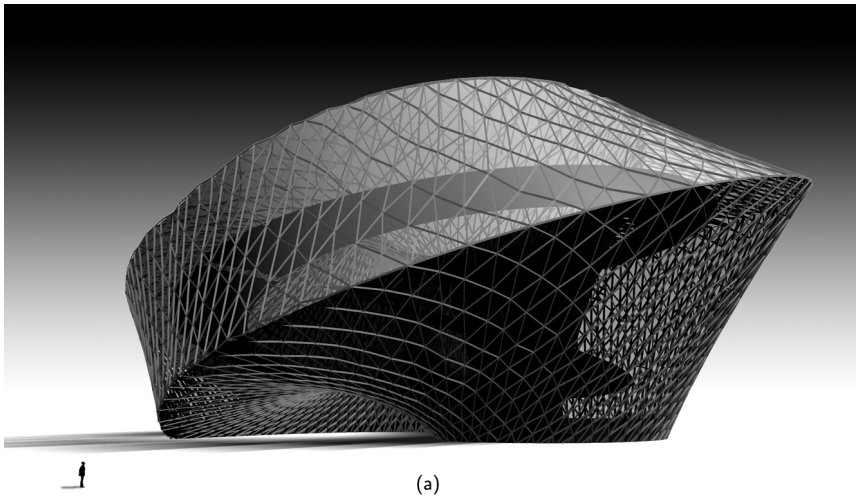
algorithm and exported the updated location information. By retaining the same topology as the original model, only the xyz coordinates needed to be updated.

For the project, the total façade area was over 20,000m^2, with a total of approximately 4500 nodes. The node pushing algorithm was used which deformed the regular façade structure as proposed by the architect to a more irregular panelling system. The façade remained constrained by the doubly curved surface geometry that could not be altered. By using this approach, we were able to reduce the total nodal displacement of the structure under permanent loads by approximately 5% after only 100 iterations. Although some nodes actually decreased in performance (see fig. 2.17), the overall result was favourable. Following this critical point, the total performance (measured using the inverse of the total deflection) began to drop off as the lengths of the members between the nodes increased.

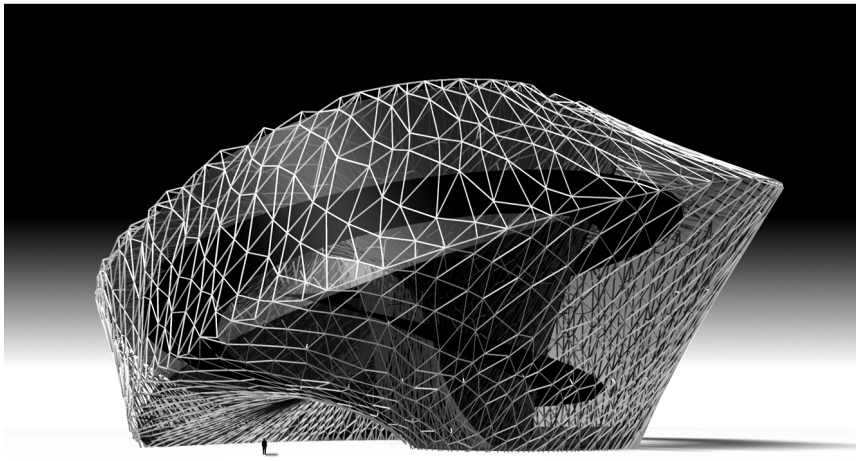
The final resulting geometry can be seen in fig. 2.18. Whilst improvements could be made structurally, the increasingly irregular pattern came at a cost to the fabrication efficiency. The façade panel sizes varied considerably more after the structural optimisation and a subsequent design review meant that the new geometry proved too costly to implement, regardless of the structural improvement. This wasn't a failure in terms of the algorithm *per se*, but rather the reality that when applying a single heuristic method such as *pushing nodes to high stress*, they do not easily allow for any other performance objectives to be included. This was the opposite problem with the node repulsion algorithm, whereby structural concerns became secondary to fabrication efficiency. As more objectives and constraints are included, the more complex the algorithm becomes.

Structural Topology Optimisation

Another structural optimisation approach used during my review period was that of Topology Optimisation (TO), in this case during the Tallinn Town Hall project [15Tth]. Similar to the principal stress method covered



(a)



(b)

Figure 2.18: Astana National Library: Initial geometry (a) and optimised (b) following node pushing

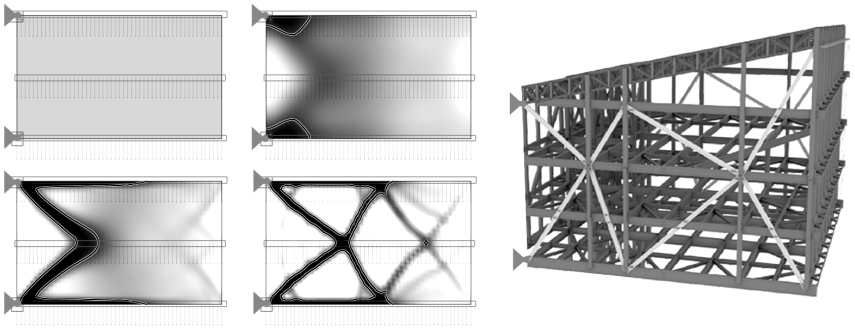


Figure 2.19: Topology Optimisation Experiment using the Homogenization method on Tallinn Town Hall

in Section 2.2.3, TO aims to find the optimal material usage for a given structural problem, achieved using an iterative method.

TO approaches generally fall into two categories: homogenisation and evolutionary methods (Bulman *et al.*, 2001). Homogenisation methods (or h-methods), work by optimising material distribution using an equality criteria that aims to distribute stress evenly throughout a given volume of material. The most popular h-method used is known as Solid Isotropic Material with Penalisation (SIMP) (Bendsoe, 2003), solved using either linear programming or optimality criteria methods (OC).

In architectural design, Michalatos and Kajima's standalone software application *Topostruct* (2011) employs homogenization methods to assist the designer in finding structurally efficient form using SIMP. On The Tallinn Town Hall [15Tth] project, an experiment was conducted using Topostruct to find a structural solution to support a 15m cantilever made from a steel frame. Although a continuous method, the results of the topology optimisation process were used to inform the placement of diagonal bracing for the steel structure. Here, a detailed h-method was used to suggest *good* configuration for the project without the final result necessarily being optimal (see fig. 2.19).

The second family of TO methods are known as evolutionary methods (or e-methods). The most well known of these is the Evolutionary Structural Optimisation (ESO) (Xie & Steven, 1997). Here, a structural heuristic is applied that removes unstressed material at each iteration, achieved by measuring the local strain energy for a given finite element and making a relative comparison. The rate of material removal is gradually reduced to zero. As well as material removal, the bi-directional variant (BESO) adds material to avoid getting stuck in local optima (Huang & Xie, 2008). As e-methods tend to be more accessible than h-methods, they have increased in popularity in the building industry for bespoke applications, for example in finding optimal structures form using fabric form-finding techniques (Bak *et al.*, 2012).

2.2.5. Summary

Heuristic methods embed some form of human knowledge in order to direct the computational process to a final result. Where there is no feedback in the process, the result is often predictable. Where there is feedback, such as incremental analysis, the result is less so. In general, heuristic methods tend to be well suited to engineering problems that can be easily stated and for which the method for getting a solution is already well known. As found the Astana National Library project, if a particular heuristic is adopted it becomes difficult to incorporate multiple objectives, such as structural *and* fabrication aspects.

As such, heuristics can tend to dominate the design process, for example on the Education City Convention Centre by Arata Isozaki. The large 'Sidra Trees' that dominate the façade were designed using the SIMP topology optimisation method, which then had to be subsequently post-rationalised by Buro Happold's SMART team. As Smith (2008, p.1) recalls:

> "...such an exercise does not account for the engineering and fabrication challenges associated with a structure of this size and complexity. The task of the design and build team was therefore to navigate the technical realities in order to realize Isozaki's architectural vision"

With regards the Sidra-tree design, a particular heuristic dominated the aesthetic and structural design and the fabrication logic followed afterwards. The learning here is that although there is no right or wrong approach to design, it is important to understand the effect of applying a particular heuristic and the potential consequences of this further down the line. It is clear that the use of a topology optimisation algorithm that dominated the early design stage meant the process of design and fabrication had to be conducted sequentially.

Heuristic methods are therefore tailor made for the engineer that loves solving a well-defined problem where there exists a optimal solution and the task is finding it using computing power. Although some of these algorithms are technically complex and have whole individual research areas devoted to them, from the point of view of collaboration in practice they are simple in that the architect typically provides a problem, the engineer develops and applies a computational approach and then returns the resulting geometry back to the architect. In essence, the engineering office acts as a black box with input and output.

Heuristic methods guide *how* a single problem is solved with the engineer's knowledge embedded in the algorithm itself. The next section looks

at a different computational approach that externalises engineering performance as measurable objectives. Within given constraints, objectives are optimised using generic search algorithms known as metaheuristics.

2.3. Metaheuristic Search Algorithms

2.3.1. Introduction

In Section 2.2 a number of heuristic methods were reviewed where knowledge of how to improve designs from an engineering perspective was embedded in the algorithm itself. Such processes are only generalisable if a similar problem arises. In this section the review moves onto *metaheuristic* algorithms that distance themselves from *how* a problem is solved.

Metaheuristic algorithms make no assumptions about the problem, and therefore require searching through a very large space of candidate solutions. Due to this non-reliance on the specific problem at hand, metaheuristic methods are generalisable and interchangeable, with their appropriateness for solving a particular issue investigated by comparing different algorithms and parameters at a higher level of abstraction. Metaheuristics are therefore more appropriate than heuristic methods when it is not clear what direction will make improvements, i.e. a poor knowledge of the possible space of solutions (Blum & Roli, 2003).

As we will see, the use of metaheuristics can be an appropriate choice for a collaborative design team, because stakeholder knowledge sits outside of the algorithm and can therefore be more easily modified than with heuristic methods (fig. 2.20). In terms of engineering, quantitative performance objectives are specified with the search algorithm using (building) analysis results in order to search for good solutions. *Constraints and parameters* typically govern the space of possible solutions that a metaheuristic search explores.

Unlike heuristic methods that embed engineering performance within the process itself, metaheuristics must by their nature conduct some sort of analysis at each iteration to understand the current state of the proposed solution. The *fitness* of a candidate solution is therefore how good it is either compared to a performance - either measured against a given metric or relative to other designs.

Other than evaluating every possible solution exhaustively, metaheuristics cannot guarantee that an optimal solution has been found without assessing the entire solution space. However, metaheuristic methods take

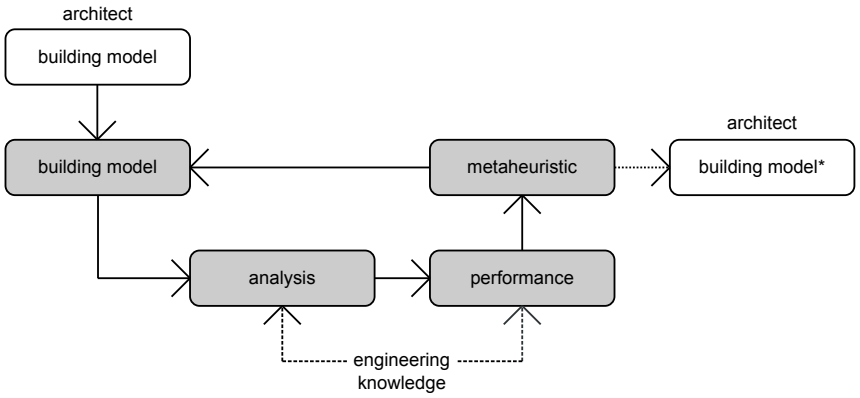


Figure 2.20: Engineering knowledge externalised as performance objectives

short-cuts to find solutions. For example, many solutions can be either ignored completely or ruled out more intelligently (depending on the algorithm), narrowing the possible number of designs to be evaluated whilst still ensuring a *good* solution is achieved in a reasonable time.

2.3.2. Metaheuristic Examples

The generality of metaheuristics mean that there are many options available to the computational designer, many with ornate natural metaphors. A comprehensive review is given by Evins (2013), although a short list of the best known approaches is as follows:

- Brute-force Search (exhaustive enumeration)
- Hill Climbing (Greedy algorithms)
- Tabu Search
- Evolutionary Algorithms (EA) *of which Genetic Algorithms are a subset*
- Simulated Annealing (SA)
- Ant Colony Optimisation (ACO)
- Particle Swarm Optimization (PSO)

The choice of metaheuristic is often a trade off between computing time and the nature of the solution space. For example, a brute force search method looks at every possible solution with an equal chance of optimality and uses no feedback in order to *steer* the search process. This means that the whole design space is covered with equal computational effort, leading to wasting computing time if a good solution is sufficient for the problem (something highly likely for building optimisation).

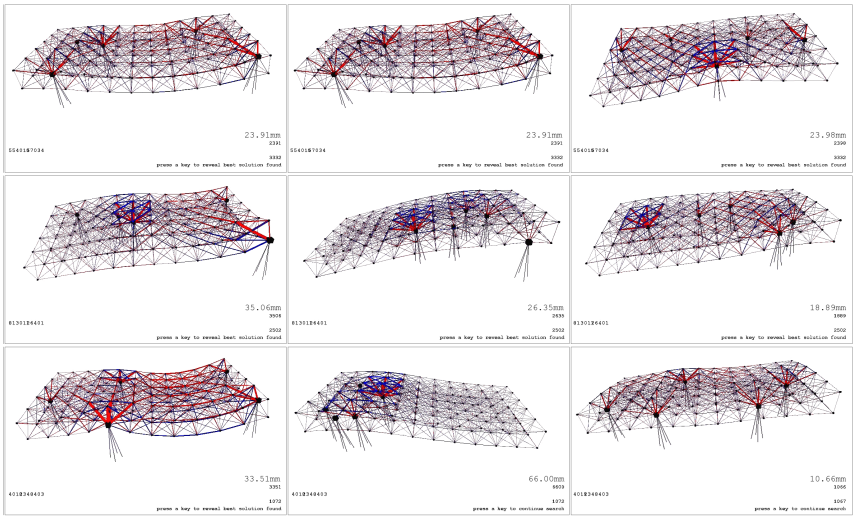


Figure 2.21: Brute-force search for finding the optimal placement of five supports. The best found after 1000 guesses is shown bottom right.

For simple problems, the use of a brute-force approach is acceptable. Brute-force search was used on the Bath House project [04Bhc] to optimise the location of columns to minimise peak deflection (fig. 2.21). Although, the total number of possible combinations was around 300 billion, the time it took to arrive at an acceptably good solution was around 1000 iterations (effectively 1000 guesses). Here, the author wrote a Java application that contained the model, analysis and metaheuristic search.

For more complex problems, brute-force methods are often inappropriate due to computing time, and hence *feedback* is required in order to direct the search process. The simplest methods that incorporate feedback are known as *greedy algorithms* (for example, hill-climbing methods). If the change makes an improvement then the solution is kept, if it makes things worse then another direction is taken. The disadvantage of greedy algorithms is that they are unable to accept worse solutions, and as a result are susceptible to get stuck in local-optima without fully appreciating the fitness landscape of the solution space. A middle ground therefore exists for metaheuristic methods that can avoid local maxima without being too computationally expensive. An comparison of some classic metaheuristic approaches is given in fig. 2.22, indicating two such approaches: simulated annealing and evolutionary algorithms.

Simulated Annealing (SA)

Simulated Annealing (SA) can avoid local optima without taking an inordinate amount of time to arrive at *good* solutions. The process begins as a random walk with gradually reducing step size and increasing propensity to move towards a local minimum (or maximum), hence the gradual cooling analogy when annealing metals. Although there is no

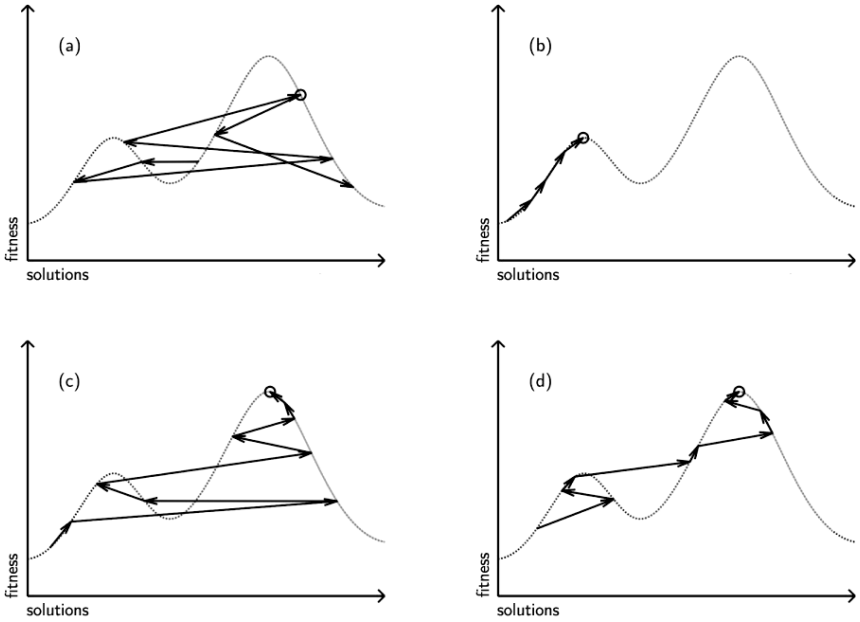


Figure 2.22: Comparison of different metaheuristic search algorithms. (a) Brute-force search, (b) Hill climbing, (c) Simulated annealing, (d) Genetic algorithm. The best-found solution is recorded at each iteration, resulting in the final design indicated by the circle.

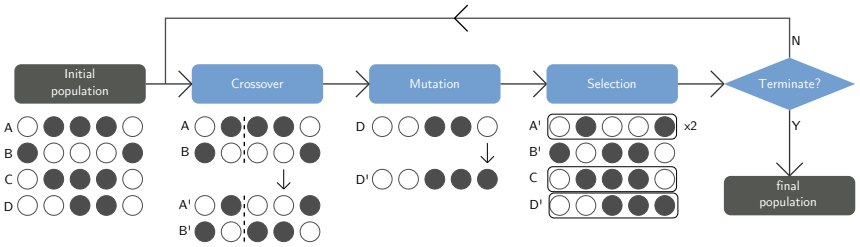


Figure 2.23: The process of genotype manipulation for a typical GA

guarantee of success, by using the right parameters to explore the design space local optima can be avoided and are therefore generally more favourable than greedy algorithms if the nature of the solution space is unknown.

Evolutionary Algorithms (EA)

Evolutionary algorithms are a class of methods inspired by biological evolution. They commonly use a genotype-phenotype mapping, aspects such as crossover and mutation operating on the genotype, with associated phenotypes assessed and retained based on a *survival of the fittest* approach. The two most commonly used EAs are Evolutionary Strategies (Rechenberg, 1973) and Genetic Algorithms (Holland, 1992). Evolutionary strategies are very similar to genetic algorithms but favour a floating point number encoding, whereas GAs tend to work with discrete bits.

A popular form of a genetic algorithm is described by Goldberg & Holland (1988). At each generation, a population of possible solutions

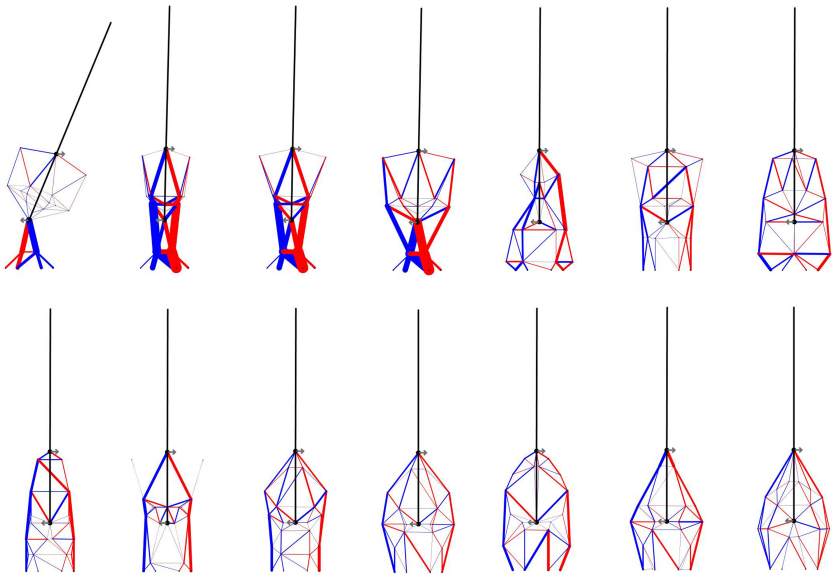


Figure 2.24: Citylife Tower Spire evolution to minimise mast deflection (note mechanism at first generation)

are evaluated, with the next generation selected using a *weighted roulette-wheel* approach (Mitchell, 1998). This new population is then subjected to gene crossover and mutation, artificially replicating a similar natural process (fig. 2.23). Manipulation occurs at the level of the genotype, and selection occurs at the level of the phenotype (i.e. the resulting organism).

As part of my review, genetic algorithms were used in order to optimise three separate projects by assessing phenotypes using structural analysis. These were The Vestas Blade Technology Centre [25Ves], The RIBA Pylon Competition entry by Ramboll [18Pyl] and The Citylife Tower Spire [27Cts] (fig. 2.24). For these projects, traditional topology optimisation methods such as those described in Section 2.2.4 were not appropriate because more flexibility was required over the modelling process to better suit other aspects of the design, including aesthetic and fabrication concerns. For example, the use of Delaunay triangulation when generating the geometry meant that no acute angles would be present in the structure, something the design team also wanted for aesthetic reasons.

2.3.3. Mapping Process

Metaheuristic algorithms rely on a mapping between the data manipulated by the algorithm itself and the resulting model that is generated and evaluated. In the simplest case, this is a *direct* mapping between the metaheuristic algorithm encoding and the geometry in the model. This was the case for example when storing the column locations on Bath House during optimisation [04Bhc] and on the Vestas Blade Technology Centre project [25Ves]. For the latter, a bit string genotype directly translated to a location of the nodes in Cartesian coordinates in the truss

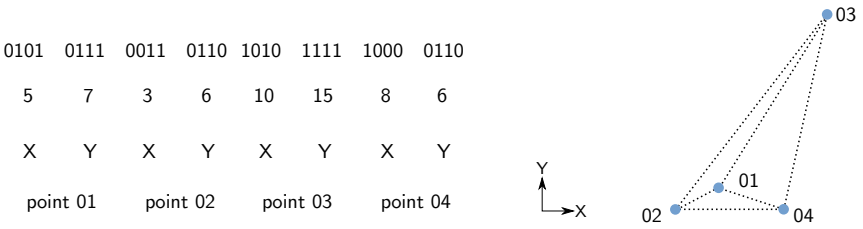


Figure 2.25: A bit string is directly mapped to point coordinates with subsequent Delaunay triangulation generating a truss structure

network (fig. 2.25). The final frame was then generated using a Delaunay triangulation from the series of points.

As well as binary or integer based strings, under some circumstances a floating point representation (FPR) is more appropriate to form the genotype for a genetic algorithm, but requires slightly different treatment in terms of crossover, mutation and selection as described by Janikow & Michalewicz (1991). Indeed, such approaches bring Genetic Algorithms closer to Evolutionary Strategies.

A natural analogy to this mapping process can be made by considering the data manipulated by the algorithm itself as the *genotype*, and the resulting form generated and evaluated as the *phenotype*. Although this terminology is most commonly used with Evolutionary Algorithms such as GAs, it is often convenient to maintain the analogy when considering the use of other search processes. The form of encoding is either *direct* (for example mapping a binary string to Cartesian coordinates as above), or *indirect* - that is, there exists an additional process in-between the genotype and phenotype during the development of form (Stanley & Miikkulainen, 2003).

The geometry created by parametric models usually have an *indirect* encoding, because an additional parametric schema exists (with its own data flow) that translates genotype into phenotype. The genotype is mapped to a set of numeric parameter values, not to the (visual) program itself. This distinction and its implications will be covered in more detail later in the context of Artificial Embryogeny (see Section 5.1.1).

2.4. Parametric Modelling

In the last section, computational processes were investigated that aim to solve problems using metaheuristics with a direct mapping between genotype and phenotype, or rather parameter and form. In this section I review an indirect mapping as a computational process in itself, made with a certain type of parametric modelling approach using dataflow programming.

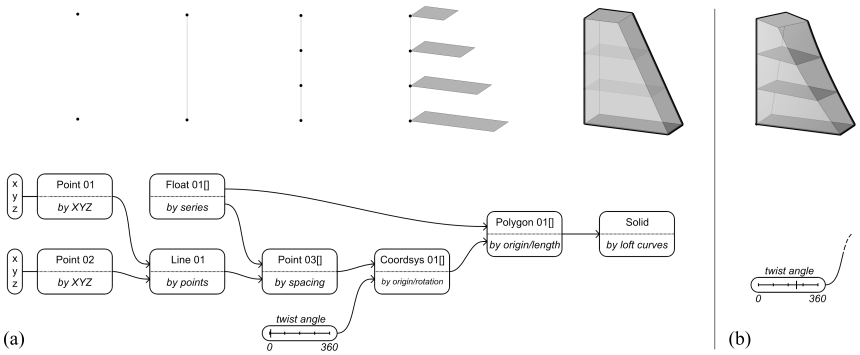


Figure 2.26: Creation of a simple building form using a directed acyclic graph (a). Manual modifications to the 'twist angle' parameter enables new design options to be explored within a user-defined range (b).

2.4.1. Dataflow Programming

Parametric modelling is now a well-established in the computational design community. Software applications such as Rhino Grasshopper by Robert McNeel & Associates allow complex ideas to be explored quickly, often able to explore designs beyond that using traditional methods of hand sketching and model making, at least in any reasonable amount of time.

Parametric models generate new geometry from input parameters (eg. numbers or existing geometry) and by forming relationships between functions that manipulate these initial inputs in some way. These relationships may not be be explicit in their causal influence for all parametric modelling software. As an example, Autodesk Revit allows the user to manipulate existing geometry, and then uses a context-driven change engine to determine which other elements need to be updated to maintain a a consistent model Autodesk (2007). Parametric modelling tools such as Grasshopper however are more explicit in how the user forms relationships between objects. A form of dataflow programming, primitive geometric functions and operations are connected with directed edges in order to form associations and describe a computational process. As such, an associative process illustrating how to construct the final model from geometric primitives can be made explicit in the form of a directed network, or Directed Acyclic Graph (DAG) (Christofides, 1975).

Figure 2.26 shows a very simple DAG-based parametric model that converts floating point numbers into a geometric form. In a collaborative context, parametric models such as this example attempt make the process of translating genotype to phenotype explicit to the design team because of its visual form. Manual manipulations to any part of this visual program can be made at any time by the user, both in terms of the numeric parameters and the choice of topology and functions that form the graph structure.

The Arts Alliance Travelling Theatre [05Aat] case study project is a

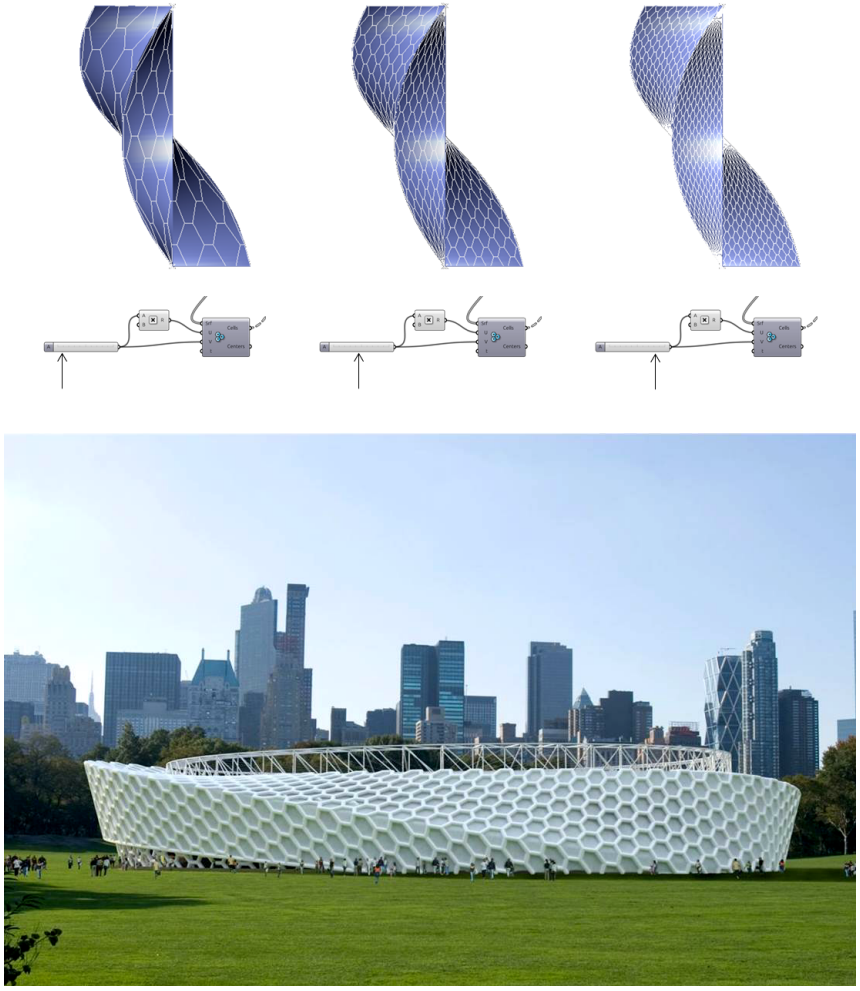


Figure 2.27: Quickly exploring different hexagonal pattern densities (a) and the final design (b) (Image courtesy of Various Architects).

simple example whereby the author assisted the architect to develop a parametric model to test out hexagonal patterns for a façade at varying densities (fig. 2.27). An adjustable numeric parameter (sometimes refered to as a *slider*) is translated into a continuous hexagonal patterning on all surfaces using a component. The architect was able to adjust the parameter dynamically whilst gaining information on the surface area of the material required to realise the design. The final design balanced the aesthetic requirement of the façade as well as the cost implications of that particular density. One only has to think about conducting this process manually (using CAD software for example) to see the advatages of using parametric models.

2.4.2. Integrating with Building Analysis

The structure of the DAG representation is then that which guides variation when parameters are adjusted. Following construction, a combination of parametric modelling and performance analysis tools allow a variety of design variations to be explored by adjusting numeric

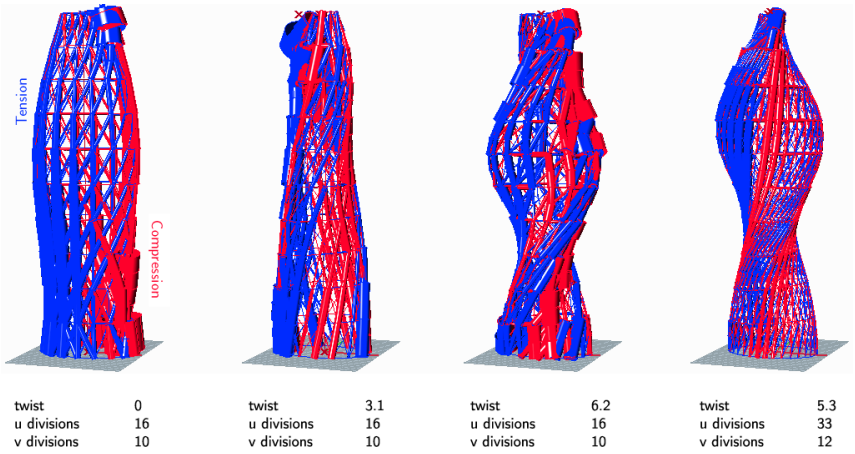


Figure 2.28: Real-time structural analysis written by the author used on the Cheongna Tower parametric model. The member radius represents axial force.

parameters with performance information communicated to the design team (Shea *et al.*, 2005). This 'slider tweaking' can occur manually or as part of an metaheuristic optimisation process guided via performance feedback (fig. 2.29), whereby the parametric model parameters become the *decision variables* of the search process.

Once a model is constructed, multiple design options be quickly be investigated. Although it can sometimes take more time to create the structure a the parametric model in comparison to traditional CAD methods, it can significantly reduce the time required for exploration, change and reuse (Aish & Woodbury, 2005). In an engineering context, Holzer *et al.* (2007, p.627) state:

> "Due to increased automation capabilities assisted by computational means and the speed with which results can be generated, structural engineers can now inform their design process through optimisation results that can be generated in parallel with structural design."

In order to get performance feedback, a parametric model can either communicate with a third party software application or integrate the analysis as part of the parametric model itself. On the Cheongna Tower case study project [03Cht], a real-time structural analysis package written by the author was used. This took an existing parametric model already constructed by the design team and performed a relaxation based analysis with the tower subjected a large imposed wind load (fig. 2.28). The model generated a hyperboloid geometry with the parameters altering the tower's twist and surface discretisation density. Different designs could be explored by adjusting these parameters and visually inspecting the structural implications in real-time.

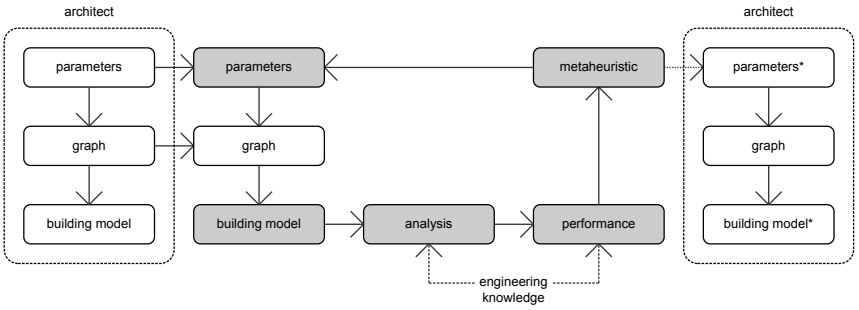


Figure 2.29: Search algorithm operating on parametric model whose graph (DAG) is already defined

2.4.3. Integrating with Metaheuristics

By setting an objective and attaching a metaheuristic algorithm an optimal design could be found automatically. Research in the built environment seems to favour the use of Evolutionary Algorithms because of their generality and their appealing relationship to natural evolution (Turrin *et al.*, 2011; Evins *et al.*, 2012), although the popularity of Simulated Annealing (SA) (See Section 2.3.2) is beginning to increase, perhaps because the search process sometimes provides a more continuous visualisation of the search struggle than with GAs.

Importantly with regards the industry, current parametric modelling tools are beginning to include metaheuristic solvers as standard allowing bi-directional graph associations, for example the Evolutionary Algorithm Galapagos for Grasshopper by Rutten (Rutten, 2013). This development of releasing *solvers to the masses* as opposed to academic circles means that their application for architectural design problems is only likely to increase in future.

Figure 2.29 shows the workflow used when an existing parametric model generated by the architect is optimised using engineering performance criteria. The concept is realised in the structure of the model, with parameters optimised using a metaheuristic algorithm. Throughout the process, the structure of the model belongs to the architect with only parameters modified by the engineer to post-rationalise the design.

After the live project had been completed on Cheongna Tower, it was decided to conduct a simple test to investigate what could have been achieved by using a metaheuristic solver in the process. The Simulated Annealing (SA) algorithm was chosen in order to optimise the tower geometry according to a fitness value. The fitness chosen was given by a combination of two performance objectives important to the engineer:

1. Minimise peak deflection of the tower when subjected to a prevailing wind load.
2. Minimise the total length of steel members, thus giving an indication of steel tonnage.

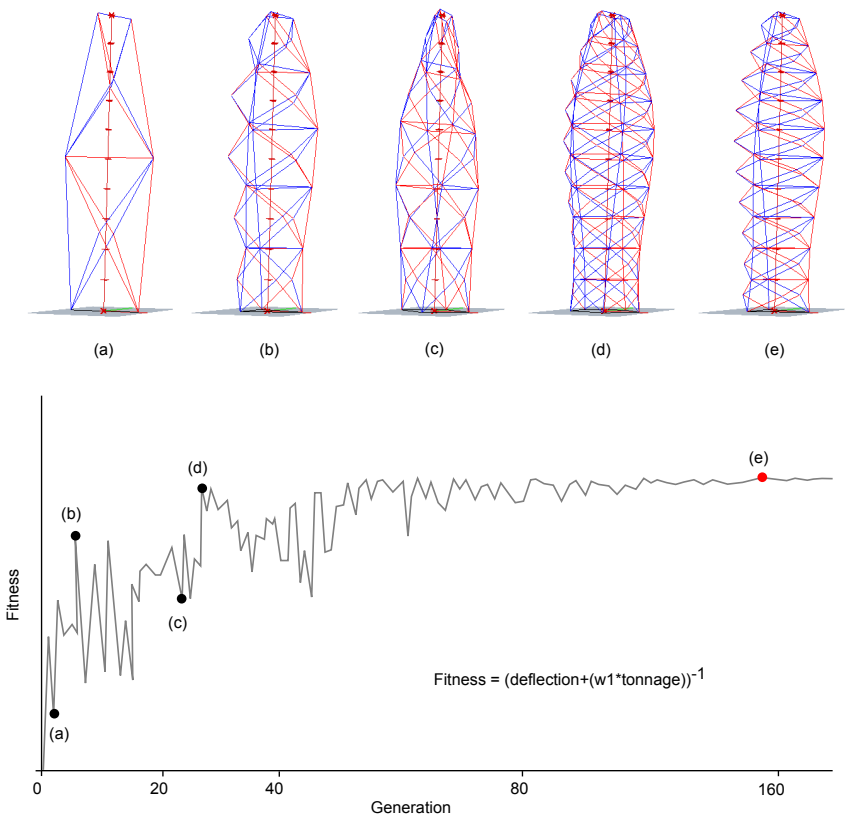


Figure 2.30: Single objective results for Cheonga Tower optimisation. Designs show deflection and tension and compressive elements

The two objectives chosen were not mutually exclusive; a denser structure would lead to greater steel tonnage but deflection would increase as a result due the exposed area of the structure picking up projected wind loads. Crucially, as a single fitness value is required, the designer must decide on the weighting between two objectives. This takes place inside the parametric model definition by simply adding a weighting multiplier to one of the objectives (see Appendix F). The decision variables were the three parameters as used on the live project (fig. 2.28).

The results are shown in fig. 2.30. Early on in the process, the SA behaves randomly and different designs are explored (a, b, c). The SA solver hits a significant local optimum (d) but was able to break free and pursue a different design direction resulting in the final best design (e) at around 160 iterations. Visual inspection found that the slight twist in the final tower design aligned the widest part of the structure in parallel with the wind load. Perhaps a less obvious finding was that the density of the member spacing in the height direction was much greater than in the horizontal direction.

The fitness axis is unit-less due to the weighting applied between different objectives. Here we see the problem of judging the relative importance when combining non-mutually exclusive objectives - i.e. which is more important than the other and how do we reconcile different units of measurement under a single objective? A higher level criteria such as

cost can assist in combining objectives, however this may not be possible in the general case. We shall see in the next section that when looking at multiple objectives, we can still produce a wide collection of *good* designs where the weightings between objectives are not set in advance.

2.5. Multi-Objective Optimisation

Due to the independence of metaheuristic algorithms, additional performance objectives can be easily added or existing ones removed. This is in contrast to heuristic methods that embed a particular direction to the computational process. In Section 2.3, case studies were shown where a single objective was to be optimised. In Section 2.4 I attempted to combine several objectives into a single fitness value, for example by using relative weightings on the Cheongna Tower Project. However, if these weightings are not decided in advance, then there still exists a method to display a collection of good designs, known as the Pareto Efficiency approach (Deb, 2001), named after the economist Vilfredo Pareto. Weightings between the performance objectives are not specified exactly, but lie within a chosen range. The process rewards only those solutions that are *non-dominated,* that is, for the case of two objectives there exists a solution that is no better solution for one objective without it penalising the other.

Figure 2.31 shows a simple example set up by the author to illustrate the concept, consisting of a series of rectangular buildings all with the same floor area that were randomly generated. A constraint on the column grid and floor to floor height was applied, with each generated design then evaluated for two performance objectives:

1. Total envelope heat loss (environmental): $f(x_1) = s^{-1}$ (where s is the building envelope surface area).

2. Total column force (structural): $f(x_2) = c^{-1}$ (where c is the cumulative force in all columns under self-weight).

The non-dominated subset of efficient solutions lies on the Pareto frontier with examples indicated (a,b,c). Whilst solution (d) lies on the front, it is heavily biased to one objective (structural performance), and through inspection this is due to it being one storey. Solution (e) however is inefficient both structurally and environmentally irrespective of the weightings applied between the objectives. Even with a randomly generated set of examples, it is easy to see how one might rule out a large proportion of designs without setting the weightings between objectives,

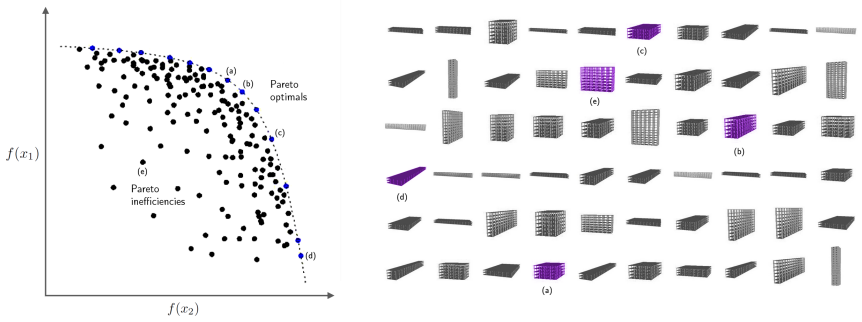


Figure 2.31: A simple multi-objective performance study showing Pareto concepts

so long of course the objectives are well-defined and will not change at a later date.

As with single-objective problems, a suitable metaheuristic algorithm is required to help arrive a optimal non-dominated set of solutions. Brute-force search is obviously the simplest approach, but again easily the most inefficient for most multi-objective problems. Again, evolutionary algorithms appear to be the most popular form of metaheuristic in use. Multi-objective evolutionary algorithms (MOGA) such as the Improved Strength Pareto Evolutionary Algorithm (SPEA-2) (Zitzler *et al.*, 2001) and the Non-dominated Sorting Genetic Algorithm II (NSGA-II) (Deb *et al.*, 2002) all make use of the set of non-dominated solutions in order to establish an elitist hierarchy for probability of selection for each new generation. This approach is similar to using the fitness to give roulette wheel weightings for a single objective problem (see Section 2.3).

As with single objective problems, the evolutionary algorithms use performance feedback from the phenotype to help direct the search process as time progresses. In a building design context, multi-objective algorithms (with more than two objectives) have been used by Rafiq & Beck (2008), and Evins *et al.* (2012). In terms of parametric models in industry, the recently released Grasshopper plug-in 'Octopus' (described by Roudsari *et al.*, 2013) uses SPEA-2, and has begun to open up multi-objective approaches to a wider audience. In such a set up, setting the problem constraints and the performance objectives becomes the task of the design team, with the machine handling the evolutionary process itself.

2.5.1. Adapting the Parametric Model

As described in Section 2.4.3, for The Cheongna Tower [03Cht] parametric model, a structural fitness score relating to peak deflection was combined with the total member length. These fitness scores were normalised and combined into a single objective *a priori*. However, if the weightings between the objectives cannot be decided upon then a Pareto

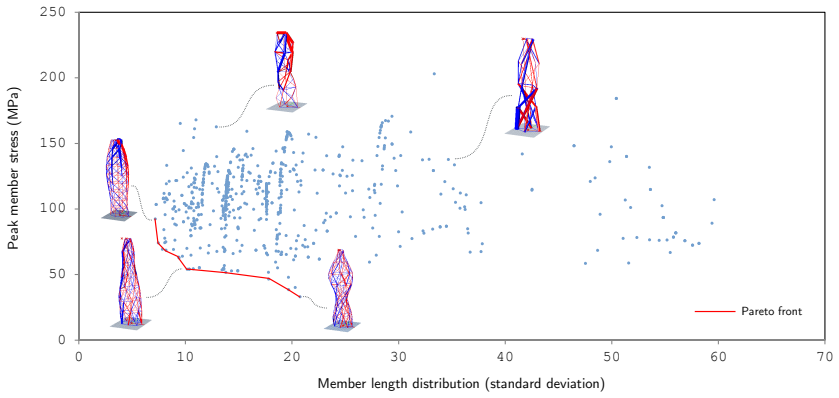


Figure 2.32: Multi-objective optimisation on Cheongna Tower

based approach will give a non-dominating set of good options to choose from at a later stage.

The short experiment for the Cheongna tower was later expanded by the author to include two performance objectives without prior determination of their relative weightings. A non-dominating set of designs could then be put forward for further consideration by the design team, essentially narrowing the search by eradicating poor designs whilst still leaving weightings open to adjustment.

For this experiment, I decided to change one of the performance objectives to be member length *distribution* as opposed to the total member length. This adjustment was simple to achieve. Instead of using a MOGA for the search, a simple brute-force was adequate to generate a set of solutions due to the relative simplicity of the problem. A non-dominating set of 10 designs were identified using this method after assessing 500 different design options (fig. 2.32).

In terms of the designs explored by the algorithm, the building typology was essentially pre-determined during the search process; i.e. all designs were constructed from twisting elliptical floors. We have seen in the previous two sections how easy it is to adapt an existing parametric model to be optimised for different performance objectives and indeed change the metaheuristic process used, however the parametric model schema that describes the set of designs *is* difficult to modify once the design team has established a concept geometry.

2.5.2. Handling Requirement Changes

The model structure gives a clear distinction between parameter constraints and objectives. It was found that modifying different objectives, changing their relative weightings and adjusting the search process itself is relatively straightforward.

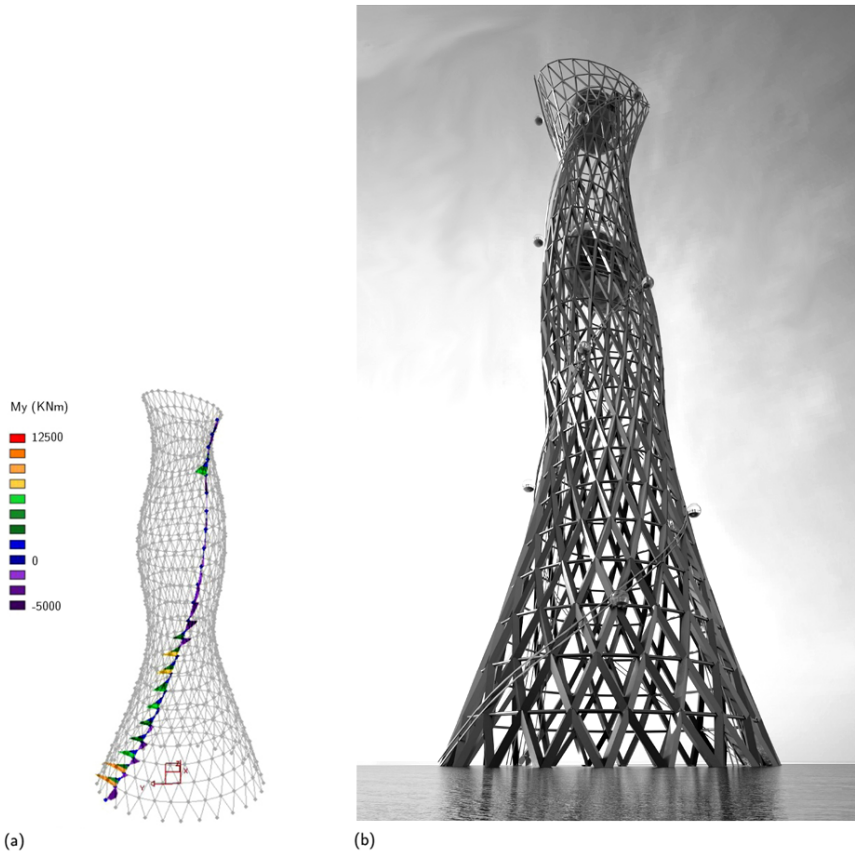


Figure 2.33: Cheongna tower final structural analysis showing high bending moments at the base due to the tower's shape (a), and the competition submission image (b) (Image courtesy of Various Architects).

In contrast, on the live project when the problem constraints changed, the parametric model was unable to deal with it. During the concept design stage, the requirements for the functional spaces increased in size after the architect located a cinema complex at the top of the tower. This meant the dead load increased relative to the wind, and proved to be an issue for the tower's elliptical shape that went into global torsion. Unfortunately, due to time constraints it was decided by the project director Stephen Melville that it would have taken too much effort to change our approach, as the conceptual idea behind the geometry was already frozen (pers. comm., June 2009) and had been agreed with the architect.

As a result of this inflexibility, for our final competition entry the members became very large towards the base to deal with the global torsion problem resulting in cost inefficiency and in the opinion of the author, having a negative aesthetic impact on the design (fig. 2.33). Better collaboration may have been possible had the modelling approach been more flexible, exploring possible design solutions outside the bounds set by the parametric schema.

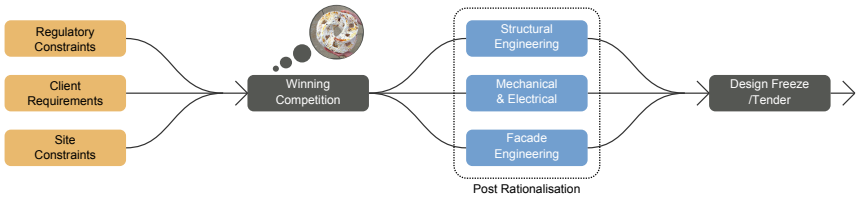


Figure 2.34: Traditional procurement for a design competition involves the sequential ordering of disciplines

2.6. Discussion

The heuristic methods in Section 2.2 impose a particular direction on the design process. Engineering knowledge is embedded in the computational approach and becomes integral to the development of form. In contrast, the use of metaheuristics mean that the engineering performance objectives are externalised, and therefore can be easily altered as projects evolve. The search process is restricted to the constraints and parameters set up in the model, which can become frozen before the design problem is properly known.

Although the computational methods used on Astana National Library [01Anl] and Cheongna Tower [03Cht] could improve the designs from an engineering perspective, in reality because the concept design approach had been decided, the resulting outcomes were only suboptimal. Affordances allowed by the parametric model were not sufficiently broad. Although we were happy as engineers to have solved a problem, this was "a false sense of having optimised a design which may be fundamentally ill conceived" as Frazer puts it (1995, p.18).

In contrast to this, the Orchid House Project [22Orc] showed that if the conceptual approach was open to manipulation and not frozen before engineers are involved, perhaps new design directions can be explored. This however must be allowed for in a computational approach geared for collaborative working.

2.6.1. Engineer as Problem Solver

Figure 2.34 shows the typical workflow for a competition process in design. Already successful architectural competitions are a common start point for the involvement of engineers, often because speculative competition work is done for no fee. Only when the client has bought into the project, do we can actually start to think about the question: *"OK, so how are we going to build it!?"* It is not realistic for architects or contractors to pay engineers for bid work that more than likely will lead to nothing. As Holzer at al. (Holzer *et al.*, 2007, p.627) mention with regards the role of the structural engineer:

> "Traditional engineering methods would see structural optimisation occur after the structural design is finalised; in this sense the divergent and convergent processes occur as consecutive steps.".

This traditional way of working has also been reflected in our approach to computational tools. As Menges and Ahlquist (2011, p.1) state:

> "It is clear that in contemporary building design practice, with very few exceptions, design computation still merely serves the purpose of extending or accelerating well established design processes."

The use of computational methods as engineers seems to have assumed the well known traditional problem-solving role, negating their true potential. As Derix (2010, p.61) comments:

> "Computation is seen as a 'problem-solving' tool to support structural, geometric, climatic, or statistical aspects of traditional work stages, not questioning the stages through the new medium.... academia deals with computation in an either completely mystified or demystified way, reinforcing the existing roles within design disciplines."

If engineers are prepared to work with architects at the front end of projects where financial risk is high, then their involvement must be short but effective. Cross-disciplinary collaboration would involve more stakeholders at the early stage of the design process, resulting in higher complexity. The key question is how computers can give new strategies in a collaborative environment where time is scarce.

2.6.2. Review of Aims & Objectives

The initial aim was to investigate how computational methods can assist the design team within collaborative practice. During the first year of participatory research on many projects it became clear that collaboration was limited due to the fact that RCD came on board after the conceptual design had occurred. This meant that the computational methods applied were well defined, and classic problem solving methods could be applied. The influence of the engineer however was relatively small, as many of the key decisions had already been made in the project.

In terms of parametric models for example, this equated to the associative model already being made, with the engineer left to 'optimise the sliders'. I realised that if I was to better understand computational design as

a collaborative exercise, I had to move further to the front end of the process in my role as a participant observer. In terms of the RIBA work stages used in the UK construction industry (Phillips, 2008), this meant actively seeking projects at stages A-C as opposed to C-E. As part of a computational process, *pre*-rationalisation as opposed to post.

3. Pre-Rationalisation

> "In print writing, the tools you generate are rhetorical; they demonstrate and convince. In computer writing, the tools you generate are processes; they simulate and decide"
>
> Alan Kay (1990)

3.1. Introduction

During the initial review period, the engineer within computational design was positioned as a problem-solver, there to improve or optimise existing architectural projects in order to make relatively small performance gains. This suited a linear way of working which broadly speaking allowed the architect to set up the problem, and the engineer solve it. However, due to the problems experienced with being involved too late on several projects, it became clear that continuing in this position for my research and indeed the position of the RCD team would not address the issue of collaboration in computational design. It was decided to purposefully move further towards the front end of the design process.

Initially, this meant working without architects, and becoming a self-contained design team working on projects from day one, developing computational approaches appropriate to the early design stage. This experience is covered in Section 3.2. A number of case studies were conducted where interactive bespoke software applications were developed and used to find form, whilst crucially *being able to alter the model constraints* during the process of design development as opposed to previous projects that relied on existing ones set out by the architect. As with Section 2.2.2, in this chapter engineering heuristics are embedded in the development of form, however the resulting designs are assessed from various design perspectives and not simply treated as problems to be solved.

In Section 3.3, I document two case study projects whereby we received a concept idea from the architect in order to *make it work*. Instead of

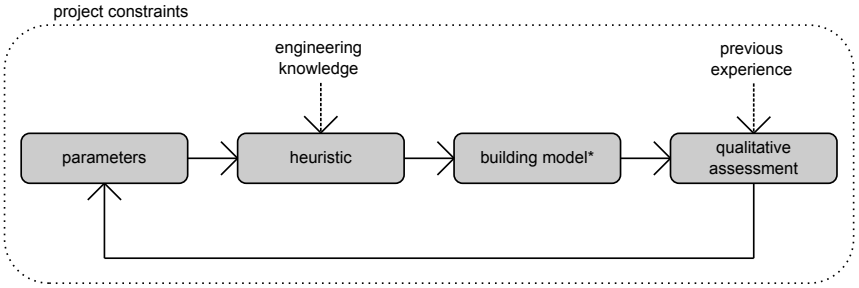


Figure 3.1: A pre-defined algorithm with embedded engineering aspects produces designs. Design exploration involves altering the parameters following qualitative assessment. As in Chapter 2, the grey shade indicates engineer control.

adopting the role of problem solver as in Chapter 2, the initial design is instead challenged and the architect asked if they may reconsider their own process of design and embed engineering principles earlier in the development of form - essentially take one step backwards to avoid problems later on in the project.

3.2. Engineering Led Design

By removing the architect completely, the engineer now acts as principal designer on the project. This simplifies the process because there are now less stakeholders with conflicting views during design development. A computational algorithm that embeds engineering knowledge can now be applied to different sets of parameters that create the model and sit within the constraints defined by the client. The engineer must meet client requirements by developing a design within the project constraints, commonly positioned external to the algorithm.

The process shown in fig. 3.1 outlines two case study projects with RCD in complete control: The London Foyer Sculpture [07Foy] and The TRADA (Timber Research and Development Association) Pavilion [24Tra]. These projects formed the key learning behind engineering led design. Both began by implementing an funicular form finding process (see Section 2.2.2), however instead of improving on an existing model, the algorithm was used as part of an *interactive* design exploration process whereby the parameters (such as support conditions, gravitational loading, etc...) were adjusted in real-time during qualitative assessment.

3.2.1. Interactive Form-Finding

In Section 2.2.2, a series of case study projects were presented whereby form-finding techniques were used in order to improve the structural performance of an existing architectural concept. In this section, engineering heuristics dominate the design approach. Such computational design processes have made something of a comeback in recent years,

Figure 3.2: The Ongreening Pavilion realised in March 2014

with designers understanding the benefits of understanding material in the development of form. Perhaps because material conservation is becoming increasingly important, there is more need to justify how architectural forms are found (Leach, 2009). This development in architectural discourse sits in contrast to deconstructivism and the unfortunate *blob* period of the 1990s where function always followed form.

Recent project examples include numerous funicular gridshells (Adriaenssens *et al.*, 2014), as well as so-called 'bending active' structures such as the ICD/ITKE 2012 Research Pavilion (Fleischmann & Menges, 2012) and the Ongreening Pavilion recently completed in 2014, for which Ramboll Computational Design were structural and geometry consultants (fig. 3.2).

One branch of form-finding is the generation of pure compression and tension shells, so-called funicular structures. In this case the structural material dominates the design process but in order to *minimise* the amount of bending in the structure due to the self-weight of the material being the dominant load case (usually concrete and brick for compressive structures). Physical hanging net models working in tension under self-weight are reversed to act in pure compression under the same loading, a principle first discovered by Robert Hooke (1675) (also described in Section 2.2.2). As the physical tension nets cannot carry bending, the resulting compressive arch and shell forms also have zero-bending under self-weight. The structure works axially with no out of plane forces, enabling a thin structural depth and material efficiency.

Antoní Gaudi's physical hanging chain models (Collins, 1963) have inspired many engineers to use similar methods in funicular form-finding including Heinz Isler (Chilton, 2010), Felix Candela (Faber, 1963) and

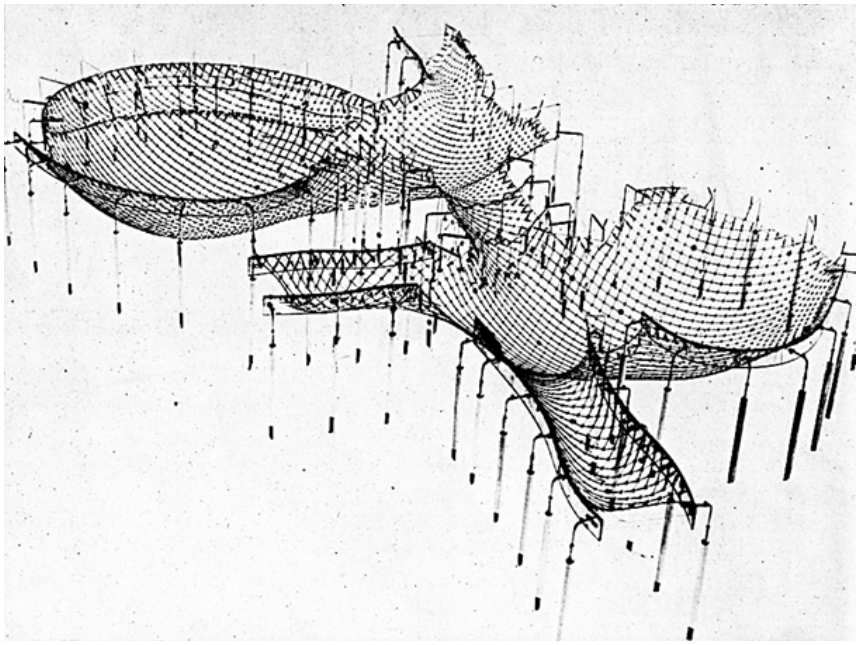


Figure 3.3: Frei Otto's Mannheim Multihalle hanging chain model. Image courtesy of the Stuttgart Institute for Lightweight Structures.

Frei Otto (Otto & Rasch, 1995) (fig. 3.3). Such methods give a real-life understanding of the behaviour of material subject to self-weight but at the cost of having to model each design option individually which can be extremely time consuming and constraining, especially if the boundary conditions are complex or the exact requirements of the design are not known.

In response to these difficulties, recent efforts have been made to re-create the playfulness and intuition of 3D physical model interaction in computer software applications for developing shells. These approaches are similar to earlier computational design software for exploring light-weight structures such as Tensyl by Buro Happold (Wakefield, 1985) and Fablon by Arup (Simmonds *et al.*, 2006).

Interactive software examples for funicular form-finding include directly mimicking physical models with stiff springs (Kilian & Ochsendorf, 2005), as well as the more recent Thrust-Network Analysis (TNA) method that combines linear optimization with projective geometry and duality theory (Block & Ochsendorf, 2007). The computational method used for the London Foyer and the TRADA Pavilion utilised a zero-length spring approach with dynamic masses (Harding & Shepherd, 2011) , a technique developed as part of this thesis. Appendix C gives more details on this method.

For the TRADA pavilion, an initial hexagonal topology of zero-length springs was established to the dimensions of the 8m x 6m site (fig. 3.4). Each node therefore consisted of 3 springs meeting at a pinned connection. Nodal mass forces relative to the local area elements of the shell were then applied perpendicular to a reference ground plane, with

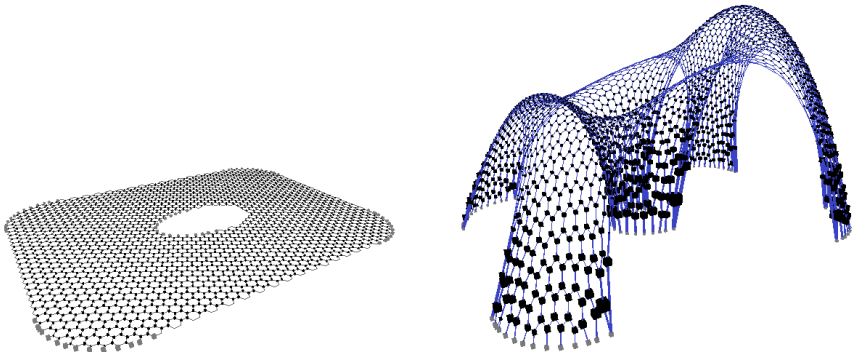


Figure 3.4: TRADA Pavilion form-finding

equilibrium found using the dynamic relaxation method (Day, 1965). The static determinacy of the hexagonal system is also useful during the form-finding process, as it enables the designer to view local forces in real time whilst the model is altered and new designs are explored. If the springs are replaced with rigid bars after equilibrium is found, the forces in the elements must remain as per the spring model in order to resist the applied nodal point loads due to the uniqueness of the solution. By reversing the direction of the point loads after finding a tensile form, a purely compressive shell with zero-bending results.

Harding and Lewis (2013) give full details on the approach used for the TRADA pavilion (a preprint is given in Appendix E).

3.2.2. Qualitative Assessment

In 1995, Frazer (1995, p.18) warned that if used unimaginatively, computers "concentrate criticism and feedback on aspects of a problem which can be quantified". Likewise, Lawson (2006, p.81) argues that "the attempt to reduce all factors to a common quantitative measure such as monetary value frequently serves only to shift the [design] problem to one of valuation". It is easy to understand this point of view, for an obsession with number and quantity are ways to make explicit and therefore easily record the performance of a particular design proposals. However, qualitative aspects of design such as what *feels* right, or aspects such as human experience, aesthetics and symbolism should be recognised even if they are not easily defined.

Without an architect, the engineer has complete control over the qualitative judgement of the design as well as the quantitative. For the TRADA pavilion, the use of timber, demountability, dimensions of the small site (8x6x4m), the small budget and the funicular form-finding heuristic algorithm itself provided the quantitative constraints. Qualitatively, the client and the RCD team wished to express what timber could achieve if used to make a funicular shell structure and hence the final design had

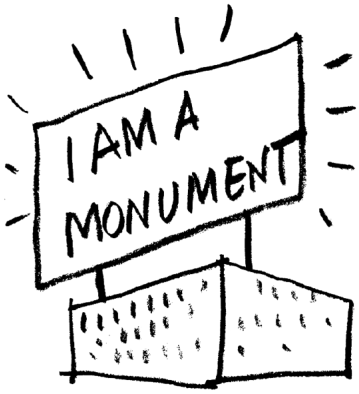


Figure 3.5: A qualitative requirement for the TRADA Pavilion (Venturi *et al.*, 1972)

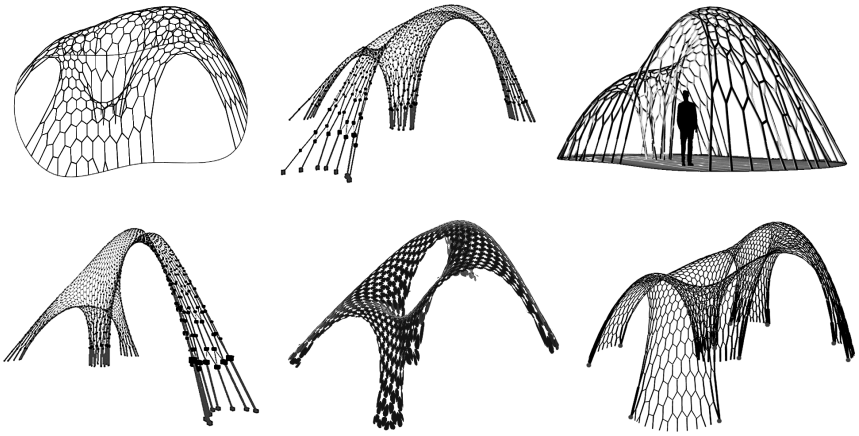


Figure 3.6: Design progression for the TRADA Pavilion. The final agreed form is shown on the bottom right.

to reflect this. On the London Foyer Sculpture [07Foy], things were even simpler because there was no client at all. For both projects, the engineer could handle the qualitative aspects because of the relative simplicity of the project.

Compared to buildings, the objectives of a pavilion that has no tangible function other than to exist and attract attention is perhaps the simplest form of architectural design. At this point, Robert Venturi's famous sketch immediately springs to mind (fig. 3.5) (Venturi *et al.*, 1972).

3.2.3. Case Study Results

Figure 3.6 shows the design progression that took place over about a month long period in our office. In dialogue with the client, various forms were explored with different support conditions and gravitational loads applied.

Reflecting on this project after the event, we can see how little design variation there was between the proposals. This was because we had embedded the form-finding heuristic in our computational approach and

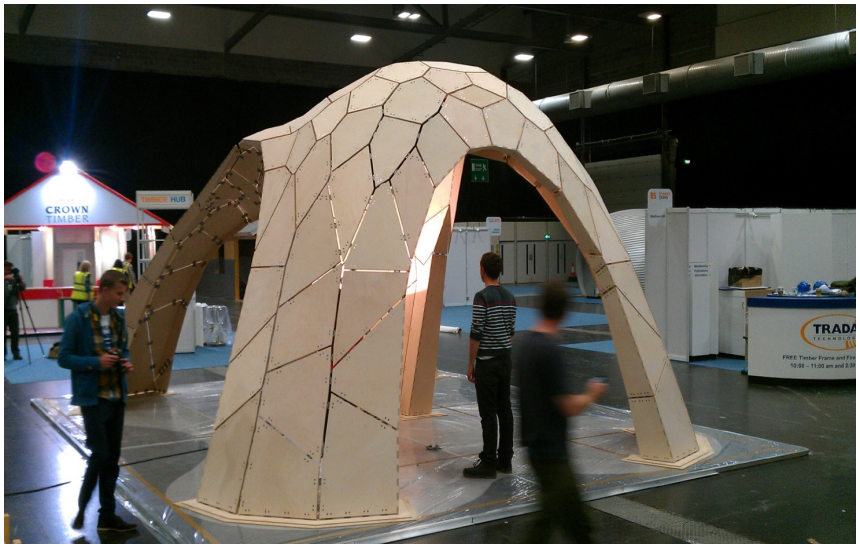


Figure 3.7: Completed TRADA Pavilion located at the Timber Expo, 2012.

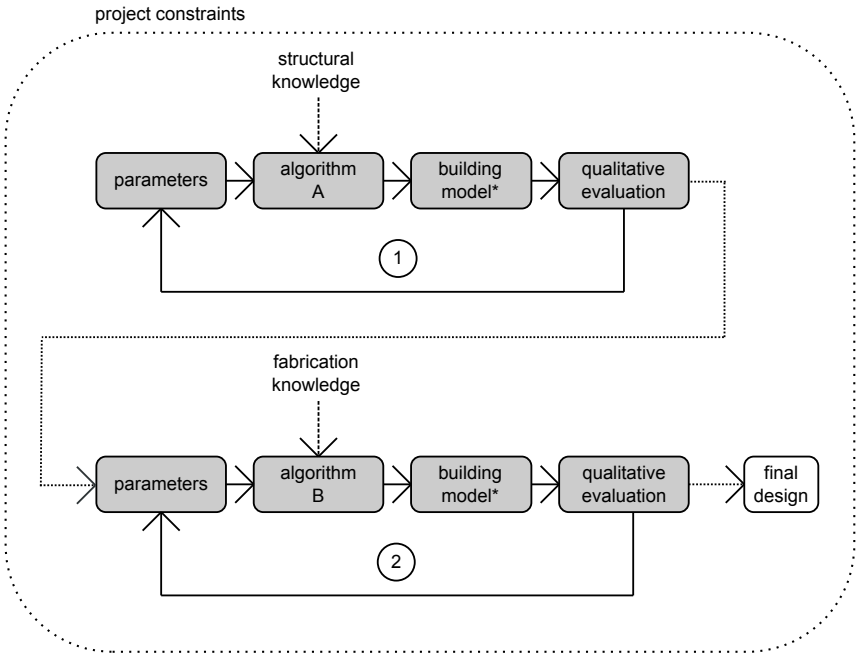


Figure 3.8: Two computational heuristics run sequentially for the development of the TRADA Pavilion

as per some of the example projects in Section 2.2, the heuristic dictated everything, essentially constraining the space of designs. Whilst this made everything simpler to have such a clear focus, it was only really possible due to the simplicity of the project and minimal requirements that remained fixed throughout. Once the surface form was found, planar re-meshing method (Cutler & Whiting, 2007) was applied in order to discretise the doubly curved form to a thin shell made from flat polygons. There were then two computational processes conducted sequentially in order to arrive at the final design (fig. 3.8).

The London Foyer Sculpture [07Foy] project was similar in that it again involved two separate sequential heuristics to generate designs. The brief from Ramboll UK itself was for a sculpture promoting the use of

Figure 3.9: Node detail on the London Foyer Sculpture

digital design to be located in the entrance space for their main UK office. The project was delivered for the launch of the Ramboll Computational Design team in 2011. More details are given in Appendix D.

This time, a form-finding process was combined with the principal curvature method (see Section 2.2.3) in order to realise the doubly curved form at low cost. This enabled flat plywood elements to be used with simple nodal connections (fig. 3.9). This process related the two key design aspects sequentially: the funicular surface geometry was found first and the discretisation/fabrication method was applied second. Clearly, one could think about beginning with the fabrication method and developing a funicular surface form as a result. Whichever we do it, because of the nature of using structural heuristics, one becomes subservient to the other unless they are embedded in the same algorithm. This however, makes the process less generalisable for use on future projects. For example, the planar re-meshing algorithm has since been used multiple times on RCD project work without any funicular form-finding being present. Likewise, the form-finding of shells does not always require such a surface discretisation.

3.2.4. Summary

As with the TRADA pavilion, the simplicity of the brief and the lack of stakeholders meant we as engineers could have full control over the design process. A clear vision of producing a structurally efficient form to fill the given volume was agreed on from the outset, and hence the heuristics used in the computational approach were able to be developed before design exploration. In this sense, there was no need to be *agile*

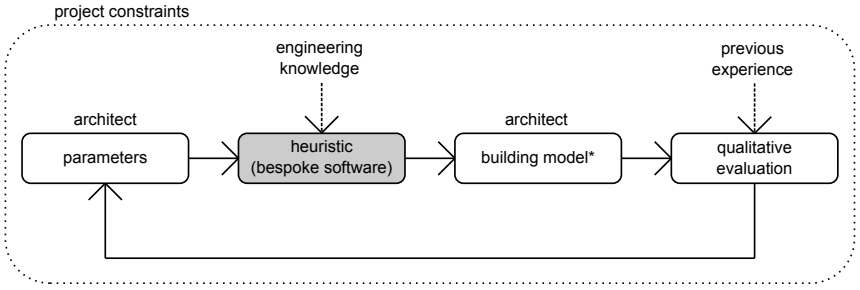


Figure 3.10: Embedded engineering algorithm is handed to architect for design exploration.

with the process chosen to generate the design which was known from the outset.

3.3. Introducing Stakeholders

Aside from these experimental projects, most work we deal with as engineers involves multiple stakeholders each with their own view of how design should progress. These include architects, multiple engineering disciplines, clients, end user clients, etc... Moving to a larger scale means working with consultants as part of a design team with their own viewpoints that are not always quantifiable when engaged in collaborative practice (Checkland, 1999).

The computational techniques developed on the experimental case study projects described in Section 3.2 produced designs with sound engineering and fabricational logic. This was because these aspects were included from day one during design development. In this Section, I detail two case study projects whereby similar heuristics developed by ourselves for bespoke projects were handed to the architect in order to attempt to integrate our knowledge into their design process at an early stage. Here, the engineer aims to constrain the architect to a particular development path, with other aspects of the design incorporated within these constraints (fig. 3.10).

3.3.1. Air Baltic Terminal Competition

The Air Baltic Terminal Competition in Riga, Latvia [13Rig] was an competition where I convinced an architect to employ a structurally dominant heuristic algorithm during the conceptual design process. Like the projects discussed in Chapter 2, our involvement as engineers initially was to *make a concept work*, in this case a continuous 400m long roof structure to be made as a timber gridshell (fig. 3.11). The geometry had been developed in Rhino, and its undulating form had a relation to the functional requirements of the terminal. Initially, the assumption was

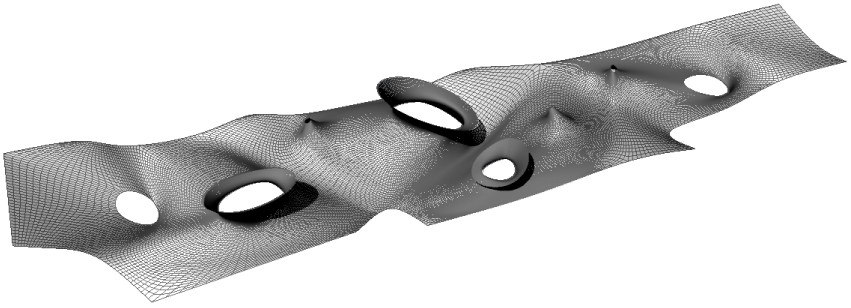


Figure 3.11: Architect's initial roof geometry proposal for The Air Baltic Terminal

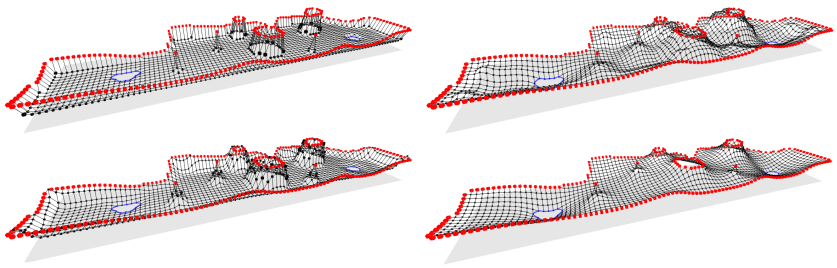


Figure 3.12: Java application with embedded engineering heuristic constraining roof design exploration

that there was no need for us to understand *why* the form was this shape as engineers and hence only the final surface geometry was sent over for us to solve and produce a structure that worked.

The roof was to be supported at circular openings, with a complex doubly curved form spanning between. Instead of following the traditional route and deploying a sophisticated algorithm to discretise this surface geometry (this was an attractive option, as solving problems and making people happy feels good!), I encouraged the architect to look again at how the geometry was constructed. I developed a bespoke Java application for them to play with to generate structurally efficient funicular forms, in this case a fully tensile gridshell. This software was similar to that developed for the pavilion structures in Section 3.2, although it also allowed the architect to investigate his own design objectives (functional, spatial, appearance etc...) within a funicular constraint. The architect was therefore *forced* to comply within an engineering constraint, albeit I had convinced them to be willingly placed in such as position.

Figure 3.12 shows several screen-shots from the application. The architect could add area loads (shown in blue) and change the gravitational field applied in order to sculpt a new doubly curved form similar to the original concept, whilst all the time it remaining funicular. The final geometry was then be exported to dxf format, sent for member sizing and analysis by the engineer and visualised by the architect. The resulting structure performed very well structurally with minimal bending (fig. 3.13), as well as being able to meet the architectural requirements of the brief.

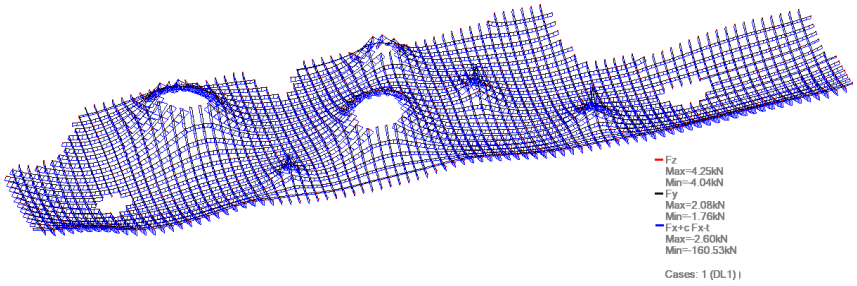


Figure 3.13: Structural analysis of the Air Baltic terminal roof (under self-weight only). The final structure carries almost all the force to the supports axially.

3.3.2. The KREOD Pavilion

The KREOD pavilion [12Kre] is a multi-use temporary grid shell structure constructed in 2012 for the London Olympics. My involvement on the project began in 2011 when the architect required assistance to make his initial scheme *work*. One approach used to improve the design structurally was shown in Section 2.2.2, although this did not consider fabrication aspects at the time that ended up becoming the dominant design driver.

At the time of my involvement, the shell form created using parametric design software had structural members both curving and twisting and meeting at complex node junction. However, previous similar structures such as the London Foyer Sculpture [07Foy] had shown that nodal connections are often much cleaner if members are aligned using the principal curvature approach (see Section 2.2.3), as well as resulting in torsion-free members.

Unfortunately, the principal curvature field for the initial design generated a structurally poor outcome and the architect was also unhappy with the visual appearance. As a consequence, I convinced the architect to see if he would be willing to change the underlying surface, if the discretisation from the principal curvature fabrication heuristic was improved. He was open to the idea, and I subsequently developed software that automated the process of visualising the principal curvature field.

The surface form was created by the architect by lofting five freeform curves. Tweaking these curves created different surfaces, for which the software revealed the lines of principal curvature in real-time to the designer (fig. 3.14). This meant that not just the form but the discretised pattern could be assessed qualitatively. As with the Orchid House Project [22Orc], it was found that small changes to the surface geometry could produce vastly different principal curvature fields. The principal curvature constraint meant the fabrication consequence for each surface were made explicit to the architect. Applying this constraint made it easier to remove the twist effect and simplify connections once the design was finalised. In essence, the workflow was similar to the Air Baltic

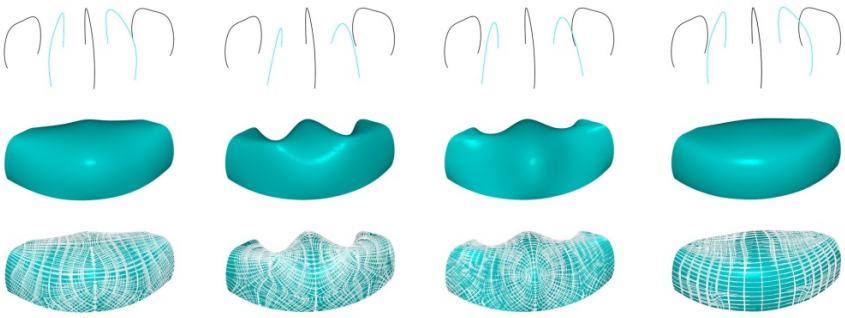


Figure 3.14: Surface/panelling designs returned by the architect on the KREOD pavilion

Terminal roof, but with a fabrication constraint forced upon the architect and not a structural one.

In the end, the method was not adopted by the architect who pursued an alternative path, still working with the RCD team, but to develop a hexagonal pattern on the surface. In effect, constraining to quads using the fabrication heuristic would have perhaps been a mistake, since the project was a success and has won multiple awards since completion in 2012.

3.3.3. Summary

The results from the two case studies were mixed. The Air Baltic Terminal project was a success in that a structural heuristic could be introduced. However, with KREOD the algorithm did little to influence the design. I later asked the architects on the Air Baltic project, Allianss Arhitektid for their opinion on the process with Ramboll (pers. comm., July 2013):

> "This [engineering] feedback is definitely not seen as a constraint. Modelling capacity is limited, but digital technologies and computational techniques can lead to discovering new and better design options."

Because the final design generated by the form-finding application was similar to their initial concept, they seemed happy to integrate it. I later asked the architect Chun Qing Li why he did not adopt the process. His comments were that although he attempted to use the algorithm and came up with some design options, they made him realise that he was still quite attached to the original form (pers. comm., 3rd March 2012). It is hard to make a general observation from these two case studies, but there is obviously a certain attachment to initial designs from architects.

Alongside the two case study projects above, a similar problem existed during the European Spallation Source (ESS) competition [30Ess], where two perfectly funicular structures were proposed to the architect (fig. 3.15). Because they completely dominated the design, the architect

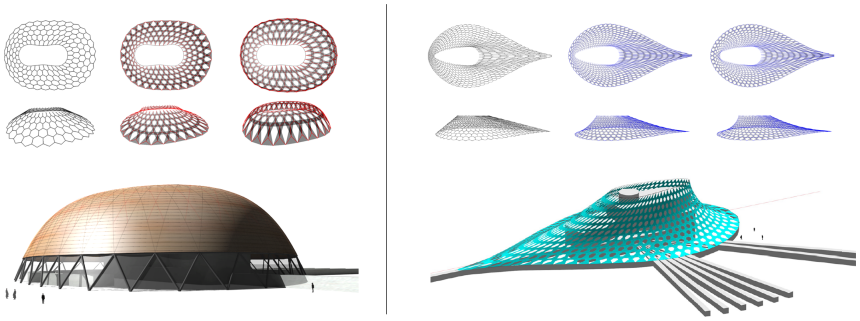


Figure 3.15: Concept designs developed by the author for compressive and tensile shells for the ESS Competition

did not adopt the ideas. It appeared from the case studies that if the engineering heuristic applied does not change the project direction too much, then it has a chance of being accepted. If not, the engineer is back to post-rationalising geometry at a later date.

3.4. Discussion

This chapter covered four case study projects in which the engineer moved to the early stage of the design process and applied a computational approach. For the first two case studies, the engineer had complete control over the design process with the only additional stakeholder being the client. Here the projects were sophisticated technologically, but relatively simple in terms of workflow due to a shared understanding of process within the team. That being said, even our own multiple engineering objectives had to be incorporated sequentially in the design process because of the computational heuristic methods used.

The latter two projects introduced stakeholders from *outside* Ramboll, and therefore added complexity to the process. With the these collaborative projects, it was not guaranteed that a particular heuristic could be introduced because of the sense of loss of control from other stakeholders.

3.4.1. Primary Generators

Architectural projects often based around a central idea formulated early on, a so-called *'primary generator'* (Lawson, 2006). If this is formed by the architect, then a computational heuristic may override everything else. For example, an agent based form generators based on circulation layouts (Puusepp, 2011). If another stakeholder such as a structural engineer formulates the central idea, they aim to embed their specialist knowledge in the computational approach (for example, structural form finding) and must attempt to convince the other stakeholders to be constrained.

In effect, this Chapter highlights exactly the same issues as in Chapter 2, but rather the architect now must follow the engineer, and not the other way around. This linearisation of the design process into a sequential workflow is just as constraining either way around. Of course, there is no right or wrong way to design and heuristic algorithms can be applied successfully in the right context. It is however crucial to understand the implication on others if a particularly biased conceptual path is chosen early in the process. Who is willing to be constrained?

3.4.2. Working Together

For larger more complex building design where multiple stakeholders have differing views of how to design, the linearisation of roles described above is often unrealistic. In comparing the development of buildings to the development of natural organisms, Coates (2010, p.165) said the following:

> "There are hardly any things in the natural world whose form/shape are the products of simple linear morphological process. On the contrary, they all seem to be the result of lots of things happening at once"

Although there may be architects that are receptive to the idea of for example a dominant structural heuristic in order to formal geometries, constraining an architect before design takes place is simply not realistic on more complex projects. This fact is one of the main advantages of Building Information Models (BIM) that aim to combine stakeholder values in one model. Plume and Mitchell (2007) noting that good holistic design is not achieved by breaking everything into parts due to their complex inter-relationship. BIM systems currently struggle to incorporate early-stage design exploration however, something discussed further in Section 4.5.1.

If we are to think holistically about the task of concept design, it appears that working together as a design team would be a beneficial place to start without one stakeholder taking over with a particular computational approach. This would go beyond collaborating on static 3D models, instead investigating collaborative processes during a wide design exploration.

3.4.3. Concept Design Stage

Moving to the start of complex projects meant working at the conceptual design stage with other stakeholders. This part of design process is

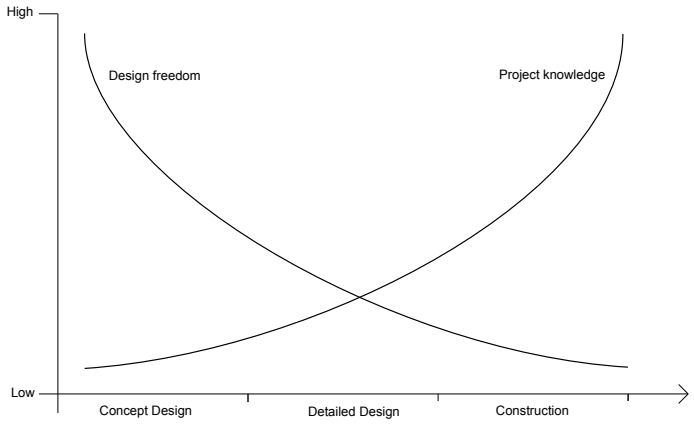


Figure 3.16: Increasing knowledge whilst maintaining freedom (Simpson *et al.*, 1998)

both where there is most design freedom, but also where the most influential decisions are made that will affect the whole process going forward (Blockley & Godfrey, 2000, p.39). It is also the time when least is known about the design 'problem' itself, with knowledge increasing as the design process progresses (Paulson, 1976; Simpson *et al.*, 1998) (see fig. 3.16).

In reality, the process of design is often a messy iterative task full of uncertainty (Schön, 1983). In architecture, Stanford Anderson (1975) for example called design not problem solving but "problem worrying", going on to say that it is impossible to state all the problems that an architectural object has to solve and that feedback from the act of design exploration constantly modifies any initial problem statement. Similarly, Menges (2012, p.4) states that: "evaluation criteria and design objectives often co-evolve with the development of a project". This is in keeping with seeing design not as a fixed process of solving client requirements, but engaging in conversation with the client and developing the brief in parallel with solutions (Lawson, 2006, p.171).

As we have seen in Section 3.2.2, even if some quantitative design drivers can be identified there exist many qualitative ones that cannot easily be defined. Some objectives remain intractable, such as the nature of spatial occupation (Perec, 1997) or the political influence of architectural form (Leach, 2001). This intractability has led to architectural design at the concept stage being likened to *wicked* problems (Rittel & Webber, 1973; Rowe, 1991).

Poor early knowledge of a wicked problem means that a response requires divergent or lateral thinking as opposed to convergent thinking (De Bono, 2010). As Lawson (2006, p.168) states: "in general, the design process needs to be more balanced and almost by definition less focussed than some polemic work may require". Design exploration should be playful and not necessarily focussed on problem solving, allowing for qualitative constraints and objectives as well as quantitative ones. One

is reminded of Charles and Ray Eames' so-called 'Solar Do-Nothing Machine' that had no function other than investigate the effect of possible project constraints and objectives through play.

When addressing wicked problems, we need to investigate computational approaches that allow multiple stakeholders to engage with the building model, open to changes in both constraints and objectives. In short, we need to be *agile* with our computational process for the complex task of early stage design, and not necessarily select a particular computational heuristic or seductive geometry and stick with it come what may.

As Rittel and Webber (1973, p.164) state, "part of the art of dealing with wicked problems is the art of not knowing too early which type of solution to apply".

3.4.4. Review of Aims & Objectives

In this chapter, computational methods have been investigated for design exploration with lose objectives, not just as problems to be optimised. It was shown that for projects where a single stakeholder or team of like minded individuals (RCD for example) is in control, a particular heuristic can be used to generate a design space. When these methods are attempted where other stakeholders are introduced with their own motivations and world views however, things can become problematic. Essentially the design process must be linearised, with one computational approach following another, and *ownership* of the design occurring sequentially.

Instead of forcing a concept geometry or computational approach onto the design team, in a collaborative environment we require computational methods that are agile and must investigate frameworks that allow for things to change and be more suitable for concept design problems. In Section 2.3, it was found that by externalising project objectives, modelling can occur independently and thus free itself from dominant heuristics, allowing for better flexibility in the future. Likewise, the use of a metaheuristic solver as part of a design exploration means externalising objectives so they can be easily changed. This finding leads directly to the next part of this thesis, a deeper investigation into the use of parametric models with metaheuristics at the conceptual design stage.

By using collaborative parametric models, many different modes of analysis both quantitative and qualitative can potentially be included at the conceptual design stage. This story then moves from an inductive methodology (i.e. letting the projects openly guide the research direction), to more of a deductive one with the statement that parametric models are

more suitable for early stage collaboration. This is investigated in the following chapter.

Part III.

Working Together

4. Early Parametric Design

> "We demand rigidly defined areas of doubt and uncertainty!"
>
> Philosopher 2, The Hitchhiker's Guide to the Galaxy

4.1. Introduction

In Chapter 3 it was shown that for collaborative projects, the complexity at the early stage increased with the number of stakeholders. The engineer could no longer take on the role of lead designer worked alongside other consultants within a design team. Instead of small bespoke software applications that address relatively simple problems by embedding a single heuristic, working on *wicked problems* means considering the use of a collaborative model in practice from the early stage of design. As discussed in Section 2.5, such models can give an independent description of a geometric process whilst allowing for multiple stakeholder input.

Potential designs can be generated that are analysed from different viewpoints and objectives. The subsequent application of a metaheuristic solvers is possible because most (if not sometimes all) engineering objectives can be quantified in terms of a fitness score. However, their relative importance between the objectives is still left to human judgement.. Crucially, the same parametric model can also be used to incorporate qualitative aspects as design drivers. In terms of workflow, externalising objectives are then more appropriate earlier in the design process because they are able to adapt, hence responding to the *wicked* nature of design.

4.2. Why Parametric Models?

In Section 2.4, we saw that a parametric model can investigate various alternative complex geometries quickly by specifying relationships

between numeric parameters and functions using an associative graph schema. Variations of the parameters will then produce variations in the form, and hence they seem an appropriate choice for early stage design where a wide exploration of building geometries are required for a healthy design search.

In this Section I will investigate in more detail the nature of using parametric design for a collaboration at the early stage with multiple stakeholders. I was fortunate that following our initial work on Astana National Library [01Anl], Bjarke Ingels Group (BIG) architects invited me to collaborate on the Escher Tower Project [16Esc], a design task in a very early stage of development and requiring engineering support.

The opportunity arose following a presentation of my thesis work to date at their Copenhagen office, formulating a shared desire to work on a collaborative project. This case study was in contrast to the early review period whereby the projects themselves guided the research direction inductively.

4.2.1. Working with the Graph

A short introduction to parametric modelling was given in Section 2.4 with regards to mapping parameters to geometric form by forming associative relationships. A particular type of parametric modelling uses a Directed Acyclic Graph (DAG) representation to build associations explicitly, with Rhino Grasshopper being the most popular parametric modelling software using this approach. The final building model is formed by computing this DAG, a process often sorted to linearise the calculation process for a single-threaded machine (Cormen *et al.*, 2001). In Grasshopper, this calculation is repeated whenever any alterations are made to the definition, thus meaning that parametric modelling is interactive - i.e. the new generated model following changes to the graph definition can be viewed in real-time.

The fact that there are no cycles in a DAG allows a hierarchy of dependency in how geometry is constructed to be formed. By using this parametric design approach, a single human user can therefore administer top-down control by modifying the associations in the graph. In essence, the act of parametric modelling using a DAG is nothing more than writing an algorithm in visual form (Gürsel, 2012).

As relatively complex models can be created much faster than with traditional computer aided design methods, the use of parametric modelling tools such as Grasshopper have become increasingly popular at the earlier stages of design. As these models further permeate the design

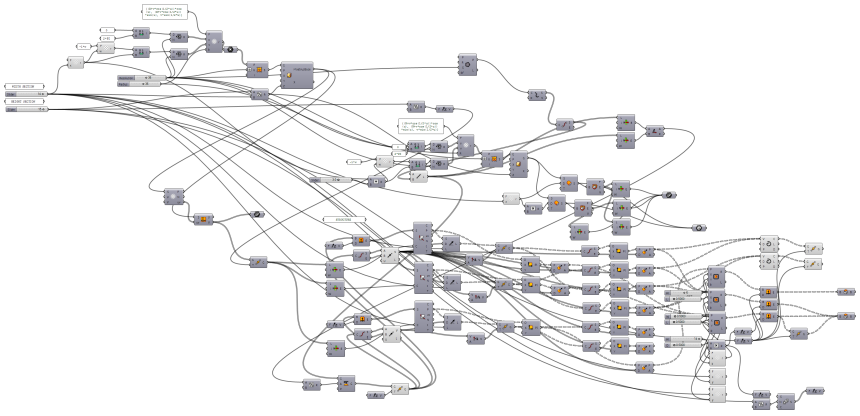


Figure 4.1: Even simple parametric models received from others can often resemble illegible spaghetti

process itself, so the term parametric design has arisen from parametric modelling, some even arguing (albeit somewhat controversially), that a distinct style known as *Parametricism* has been created as a result (Schumacher, 2009). Unfortunately, this term is meant to include aspects of complexity and feedback for which parametric models do not traditionally allow, serving only to compound the confusion over the term 'parametric' in a design context as discussed in Section 1.1.3. Besides the parametricism-as-a-style debate that will not be discussed further here, it is certainly the case that adopting parametric modelling tools such as Grasshopper at any stage of a project will have an implication on the process of design, not least because unlike CAD systems, an historical artifact of *how* the final geometry is formed is recorded, not simply the final geometry itself.

4.2.2. A Collaborative Framework

The top-down nature of parametric design means that it can often be a solitary exercise (Harding *et al.*, 2013). Attempts to work with collaborative parametric models have been proposed however and implemented in practice successfully. In such cases, the *legibility* of a DAG representation has been key to enabling other consultants to understand and interact with an existing definition. To this end, well structured parametric models can enable all stakeholders to develop an understanding and engage with the graph, as opposed to it resembling a tangled mess of spaghetti, making it hard for team members to follow geometric relationships that may have been created by others.

This was an issue when initially receiving a parametric model definition on Astana National Library [01Anl] (fig. 4.1). As a response, good structuring principles at different hierarchical scales inspired by modular programming can assist with collaboration (Davis *et al.*, 2011c) although the reality of early-stage practice does not always allow for this. The advent of *clusters* in recent versions of Grasshopper (at the time of

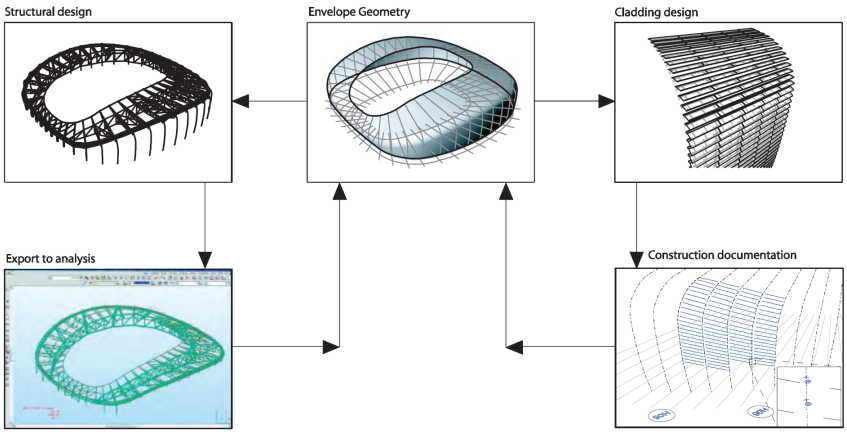


Figure 4.2: Collaborative model used on the Aviva Stadium Project. Image: (Shepherd *et al.*, 2011)

writing) should make it easier for design teams to adopt such a modular approach in the future.

A parametric model provides a collaborative framework of geometric process for others to understand in different ways. As Williams and Hudson (2011, p.27) state:

> "Understanding involves producing a simple, coherent mental model or framework to which new pieces of information can be appended, including design decisions."

Shepherd *et al.* (2011) describe the successful use of parametric models for collaboration on a stadium design. It should be noted however that this collaboration was beyond the conceptual design stage with a 'bowl' stadium typology already established. Here, individual parts of the whole model could be used by multiple stakeholders and fed-back into the main model once completed (fig. 4.2), using modular principles similar to that suggested by Davis *et al.* (2011b).

Holzer (2010) explains how decision support systems have been developed that link parametric models and BIM (Building Information Modelling) and performative analysis using the *DesignLink* framework for connecting several building disciplines.

In these examples, an explicit representation of process means that all stakeholders can have access to how the building geometry and therefore additional elements from other consultants can be added in a collaborative model. The graph's explicit representation of geometric process is important in facilitating collaborative understanding. In this sense, the parametric model provides a common cognitive artifact (Norman, 1990) between all stakeholders.

The level of interaction with a parametric model is addressed by Aish & Woodbury (2005), where they argue that multiple modes of representation - not just the graph - are required because different people see the

same task in different ways using their own mental models of perception or *world views* (Johnson-Laird, 1983). This feature went on to become one of the main features of Generative Components.

4.3. Design Exploration

As we saw in Section 3.4.3, the most important decisions in the design process are made at the start of any project, and so tools which assist in decision support at this early stage are of assistance to design teams. It was stated in Section 4.2.1 that a recent rise in performance analysis tools that can be used within the modelling program itself are becoming more prevalent as computing power increases. Structural analysis using the plug-in Karamba (Preisinger, 2013) and environmental analysis using Ladybug (Roudsari *et al.*, 2013) are two such examples for Grasshopper. As computational analysis becomes faster for visualising performance objectives, these are beginning to move to the early stage of the design process.

It was shown in Chapter 2 that imposing a particular computational heuristic too early in the project will constrain the design problem before it is properly known. In this sense, it is better to externalise engineering performance objectives as with our previous case study examples using a metaheuristic approach shown in Section 2.4.3. This way, no particular design objective rules over any other early on in the project - although at the cost of additional complexity. Lawson (2006, p.63) indicates the risks inherent with prioritising one aspect before another when assessing design options:

1. The various performance criteria are not likely to be equally important, so some weighting system is needed.

2. Performance against some of the criteria can be easily measured while in other cases this is more a matter of subjective judgement.

3. Finally, we then have the problem of combining these judgements together into some overall assessment.

In relation to point 1, the design team will need to establish these criteria, their relative weightings and the method of measurement. In addition to manual methods, we saw in Section 2.5 that a multi-objective optimisation approach can leave weightings undecided whilst still excluding poor (non-dominated) design solutions.

With regards point 2, in Section 3.2.2 it was argued that any computational strategy should respect that not everything can be measured with

a 'fitness' score. Some aspects will be qualitative and will change in time. As an example, what if the most perfectly efficient building shape is a swastika in plan, is this still a suitable option if you didn't factor Google Earth into your optimisation approach from day one?

Qualitative assessments from multiple stakeholders cannot be averaged out unless one aspires to a design by committee approach. Clearly, being able to predict how other stakeholders such as clients and architects will behave on a project is impossible, and this fact needs to be respected in any computational approach, especially those used in combination with other design exploration techniques such as sketching and model making. Building design is not an engineering problem to be solved from day one, even if it may become so at a later date.

Finally, in relation to point 3, stakeholders should not attempt to impose which design aspects are the most important too early. This process should instead be formed throughout each project through discussion. To this extent, for the early stage engineers should suggest *but not impose* engineering aspects or risk alienating themselves. A good analogy in this regard is the concept of smart power meters that constantly update the user as to the impact of using power passively. Behaviour change is not forced but rather encouraged by making users better aware of their actions, otherwise known as *nudge economics* (Thaler & Sunstein, 2008).

4.4. Implementation

As a result of working with Bjarke Ingels Group (BIG) Architects during my review period, an opportunity arose in May 2011 to join them at the early stages of The Escher Tower Project [16Esc]. BIG were open to early stage multi-disciplinary design and hence were the perfect partners for this study due to our already established relationship from the Astana National Library project [01Anl].

I acted as a participant observer during the project, aiming to learn about the challenges that face the design team in the early stages when attempting to use a parametric modelling approach. Located in Copenhagen, the proposed tower was to be mixed use (commercial/residential). The architects had already attempted some initial formal studies using foam models (fig. 4.3). These physical models were perfect for getting quick formal massing studies on the site, enabling the architect to get a feel for the impact of various geometries and their suitability in context.

The physical models and sketches offered a good sense of play and exploration, crucial to the early design stage where all ideas should

Figure 4.3: Foam models allowing exploration of different design typologies on the Escher Tower project.

be investigated, however it soon became clear that there was no way to extract any quantitative information for each design (for example, usable floor area or environmental performance), broadly speaking what could be described as the aspects of design more related to engineering. Establishing relative comparisons between different building typologies therefore relied on only qualitative judgement alone and experience from previous work. This project then offered an opportunity to how we could be involved in the early stages and offer collaborative support using computational methods.

4.4.1. Constraints and Objectives

At an initial design team meeting at the architect's Copenhagen office in April 2011, an initial set of constraints and objectives were formulated for the tower (fig. 4.4). During these discussions, it became clear that to enable healthy design exploration, as few constraints as possible should be applied. Site dimensions, local infrastructure and local regulations formed obvious boundaries simply because these were assumed as non-modifiable. One additional client constraint was the required floor area breakdown, 12,000m^2 office and 3,000m^2 residential. Certainly, no structural heuristics were constraints from day one like those described in Chapter 3. In terms of objectives, these were broken down into quantitative and qualitative categories, with no one aspect had dominance over any other initially. No two objectives were assumed to be mutually exclusive.

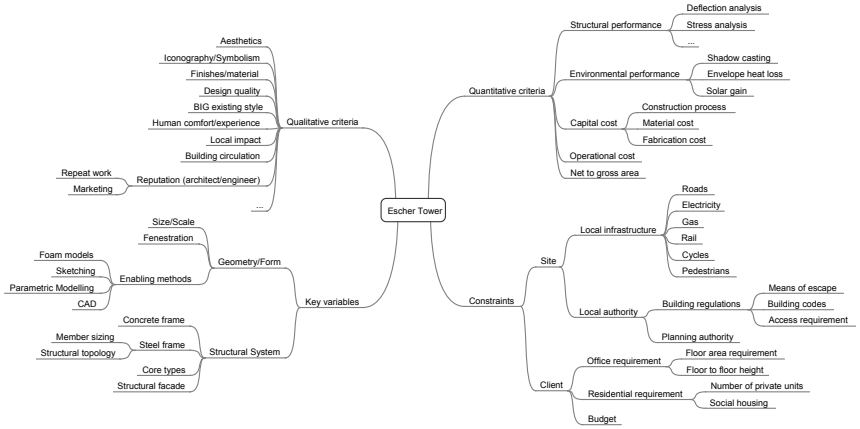


Figure 4.4: Constraints and objectives search

4.4.2. Performance Evaluation

In order to obtain some quantitative performance data for each foam model and relay to the wider design team, a computer modelling and analysis process was required. By working with our Building Physics team at Ramboll, I began by testing each option against various performance objectives in the short space of time the concept stage allowed. The process required building a separate parametric model for each building typology, with each evaluated for building performance by using either direct information (such as floor area) from the parametric model itself, or using simulation, both within grasshopper and through exporting a model to another software package (fig. 4.5).

The first building massing model explored was a series of twisted boxes that made up the tower, with a parameter controlling the angle of twist between each subsequent box. By adjusting the parameter, real-time feedback was given by the analysis package. For a single building typology, this set up was therefore similar to the Cheongna Tower project (Section 2.5.1).

In the short time available, we managed to explore three different building typologies by varying model parameters. In reality, to test every one of the BIG options that were made as foam models would have taken far too long, and it proved to be impossible to provide quantitative performance data for more than 3 typologies. Building *many* different parametric models and not just adjusting numeric parameters was required at this early stage and I simply didn't have time to do it, nor did the architect.

4.5. Observations

Once the models were constructed, the environmental and structural analysis provided quick results for parameter changes - indeed the software

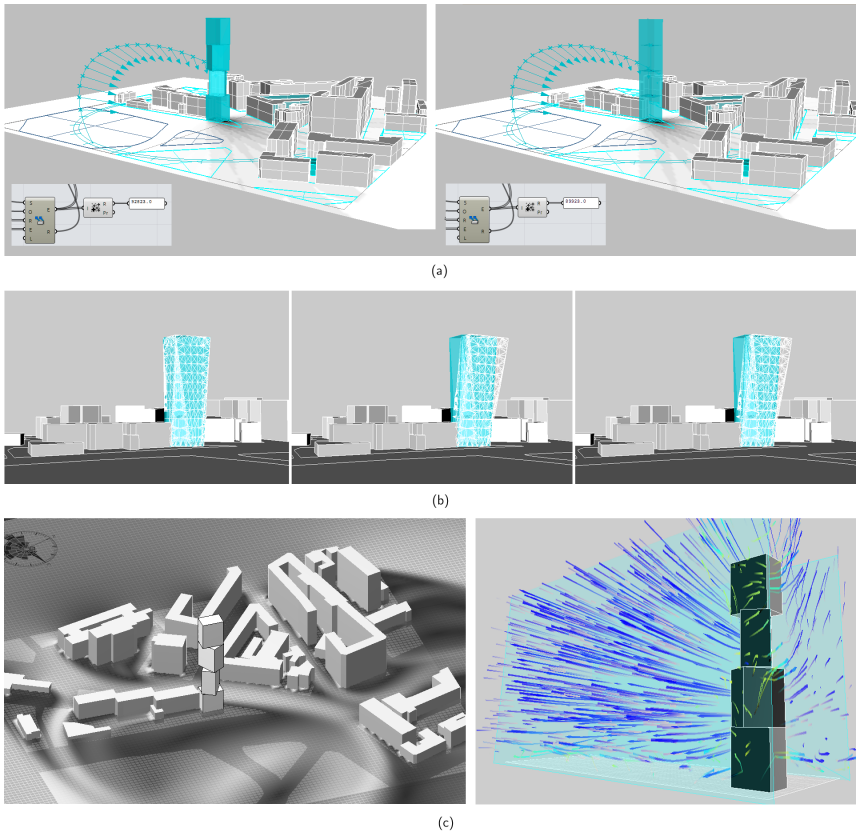


Figure 4.5: Early stage performance feedback. Tweaking a form with real-time shadow casting analysis on neighbouring buildings (a). Structural façade deflection test for a prevailing wind load (b). Using Autodesk Vasari's CFD environment (c)

was fully capable of giving a good comparative measure of performance if used correctly. However, it became clear that no modelling software was available that could adequately generate models for each design within a short period of time.

Although by using numeric parameters some variations to each design could be explored, for each individual building type a completely new parametric model needed to be built *top-down* to adequately represent each design option – something that was impractical at the concept design stage even though the parametric models required were relatively small in size. In the end, the number of options had to be narrowed down significantly before any structural analysis could be undertaken meaning that potentially good design directions were missed.

The key learning points from this exercise were the following:

- BIG used foam models in order to be able to quickly explore design directions, but they lacked the quantitative analysis that may assist in deciding which design path to develop further.

- In contrast, the engineers had a good knowledge of which building performance objectives to analyse, but the computer models were not flexible enough to explore a wide variety of form.

4.5.1. Concept Design Exploration

We have seen in Section 3.4.3 that the conceptual design stage is a wicked problem. However, this fuzzy nature of the early stage appears to run counter to the reality of parametric modelling in practice. Parametric models develop incrementally, and are constantly refined during a project (Peters, 2007). Woodbury (2010) describes this as "erase, edit, relate and repair". Every addition makes the graph structure more complicated, so the model's flexibility and potential to adapt to changing constraints and requirements gradually reduces. This problem is compounded if the graph structure is poorly organised.

As discussed in Section 4.2.2, if put together quickly and without legibility for others in mind, the graph can often resemble a tangle of spaghetti, making it hard to follow geometric relationships and thus interact with. In terms of design exploration, we are left to merely tweaking sliders so that nothing more than fine tuning can take place (Davis *et al.*, 2011b). As Aish and Woodbury acknowledge (2005, p.11):

> "Nothing can be created in a parametric system for which a designer has not explicitly externalised the relevant conceptual and constructive structure. This runs counter to the often-deliberate cultivation of ambiguity that appears to be part of the healthy design process"

This inflexibility is not necessarily a bad thing if the building typology is already agreed upon - for example the rotating boxes concept (fig. 4.5) could have been easily explored and real-time results visualised as decision support. Parameter changes are therefore limited to a single topological graph structure for the parametric model. i.e. the components and their relationship cannot change without top-down effort. Contrast this to the speed in generating completely different solutions using hand sketching or physical working models, and we see why they are so popular at the concept stage where the design process jumps between disparate ideas.

This isn't to say we should ditch our machines and return to the sketch. Firstly, parametric models can generate designs beyond models and sketches. And, as I found on Escher Tower, extracting quantitative performance is often impossible with sketches alone. This ability to extract quantitative data does tempt the design team to begin with computer models early on in the process. For example, clients will often specify usable floor area requirements, and so having this information available for possible designs is of benefit.

Turning to the computer from day one has been encouraged by developments in Building Information Modelling (BIM) and performance

based design (Eastman, 2009), that champion the parametric model as the suitable vehicle for design exploration at the concept design stage, for example being the combination of Dynamo (parametric modelling) and Autodesk Vasari's real-time performance analysis tools (Boeykens, 2012). However the BIM community has run into the same limitations. As Heumann (2012) states:

> "Decoupling a model from its variables, breaking it out of its fixed dimensions and dependencies, may be antithetical to the efficiencies of BIM but may in fact be essential to the creative process of design."

Indeed, the level of detail required with building information models and their resulting inflexibility makes more suited to the later stages of the design process (Cavieres *et al.*, 2011). Likewise, the topological inflexibility of a parametric model goes against healthy design exploration that helps form the design 'problem' itself as well as its solution. As Richens (2011) states:

> "There are many ways to construct and parametrise a model, and the choice you make encapsulates a decision as to which kinds of future change will be facilitated and which not"

Building models can become over-parametrised, restricted to a single design 'body-plan'. Coates (2010, p.105) gives a very good example:

> "The great failing of these kinds of algorithms is that while they are good at exploring the design space defined by their parameters, they cannot transcend them. For example, genetic algorithms can be used to design the hull of a sailing boat, with parameters for curvature and a fitness function of minimising drag, but such an algorithm would never come up with a catamaran."

4.5.2. A Fixed Body-Plan

In nature, the body-plan of an organism is how its main parts are laid out during the development of form (Wolpert, 1991). A direct analogy to building geometry is easy to make. With parametric design systems, the associative DAG structure can be seen as the 'body-plan' of the building, i.e. how it is formed from primitive component parts. We saw in Section 4.2.1 that parametric models have an encoding between parameter and form that include elements such as repeatable parts, similar to those found in nature. These have also been replicated in artificial systems,

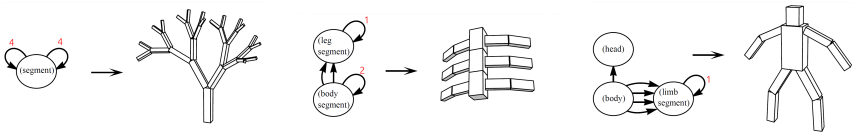


Figure 4.6: Body morphology of three creatures. The author has added the recursion depth values. Source: (Sims, 1994)

such as Richard Dawkins' Biomorphs (1986) and Karl Sims' evolving creatures (1994).

Sims used directed graphs in order to represent specific body parts with repeatable elements (fig. 4.6) and therefore provided an explicit representation of the body morphology. Sims' graphs included simple recursive elements, and while it is worth noting that traditional parametric models do not traditionally deal with cycles, recursive structures can be explicitly represented because no additional information is required to explain the entire process. A further discussion on this with regards formal grammar rewriting can be found in Section 6.2.3.

Indeed, plug-ins to include recursive structures in parametric models and hence increase their functionality have already been developed, for example the 'Hoopsnake' component for Grasshopper by Chatzikonstantinou (2013). It is now possible to integrate simple recursion into a graph definition, replicating the capability of Sims' networks made almost twenty years earlier.

Unlike Sims' networks that evolve alternative body plans, we have seen that parametric models are by their nature are combinatorially inflexible in any search process. By becoming locked-into a parametric structure too early on in the design process, exploration becomes restricted to certain building typologies instead of considering topological variations to the body-plan. The philosopher Manuel DeLanda (2002) identified this potential problem in the use of genetic algorithms in architecture, encouraging designers to "think topologically" in order to explore a space rich enough so that all possibilities can be initially considered by the designer.

4.5.3. Design development

During design development, the topological inflexibility of parametric models means design teams gradually become *locked-in*, with any large changes to the requirements brief or constraints resulting in the time-consuming rebuild of new definitions from scratch. This experience was first documented by Burry (1996) when modelling part of La Sagrada Família. In this case, a particular conoid surface geometry was not fit for purpose, whereas paraboloids would have possibly offered an alternative. However, in order to investigate their use there was no

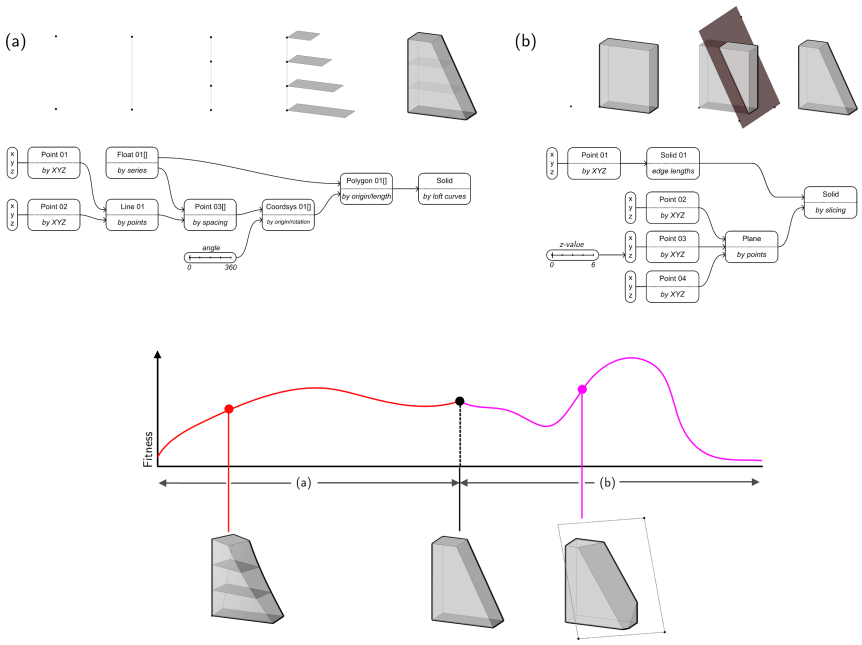


Figure 4.7: For an identical final form, a different graph structure representing development leads to exploration of different design domains (a) & (b)

option other than to disassemble the model and restart from scratch. A similar experience was found by Holzer *et al.* (2007, p.639) on a stadium roof project "whereby changes required by the design team were of such a disruptive nature that the parametric model schema could not cope with them". This is clearly a wider problem in industry and not just through my own personal experience at Ramboll.

Often, a given design by itself will not provide obvious evidence for this inflexibility. For example, the relationship between the graph representation and the geometric model is many-to-one, that is, we can create two graph structures that both produce an identical geometric model. A simple example is shown in fig. 4.7 where a different graph generates exactly the same geometry as in our earlier example shown in Section 2.4 but in a different way. Instead of lofting through a series of scaled rectangles, a cube is sliced with a plane.

Different graph structures make explicit different design intentions or investigations. This means that although the initial form may be identical, the way it is represented in the graph will influence its future development. It is the explicit structural constraint of the *component functions and graph topology* that guides subsequent variation that may lead to a sub-optimal design. Choose the wrong structure too early, and you limit your design exploration before you have properly explored the space of possible designs as found with Escher Tower, or not be able to adapt to changing requirements throughout at project as found on the Astana National Library and Cheongna Tower.

4.5.4. Possible Solutions

One response to the inflexibility of parametric models is to make the graph representation more legible and therefore easier to return to and make adaptations. Davis et al. (2011b) argue that by using better structuring using principles from modular programming, enhances the cognition of the parametric schema by others by working at different scales of hierarchy (Simon, 1962). Conventional strategies of good source code management and documentation can also help, although in reality there is often little time to prepare such information within a concept design environment.

Another approach to speeding up the parametric design process is by identifying common 'design patterns' (Woodbury *et al.*, 2007), and then re-using them on multiple projects. These become generic snippets of visual program that solve common problems, rather like the copy and paste of source code. Although certainly attractive to the time-pressed designer, the encouragement of design patterns runs somewhat counter to the idea of allowing for new ideas (unless they are strictly used to solve *well defined* problems). A popular example of this effect is the overuse of Voronoi diagrams in computational design. As Moussavi states (2011):

> "parametric design as a style disposing itself of the restraints of external parameters and promotes the autonomy of architectural forms, while it cannot advance beyond new ways of shaping matter to produce unexpected spaces."

One can easily become a slave to the pre-compiled tool because this is the easiest thing to turn to when faced with the challenge of design. Computers run the risk of distorting the design process to fit within the limitations of the most easily available program (Frazer, 1995).

Another approach is to abandon the directed nature of parametric models, and shift instead to a constraint based logic programming approach. Logic programming allows the user to set up undirected associations between objects and allow the machine to interpret (using a metaheuristic solver or otherwise) the dataflow through the graph. This approach was first investigated by Swinson in the 1980s (1982) and led to further academic developments in the 1990s for which a thorough review is provided by Fudos (1995). Unlike parametric models, the directed flow of the graph need not be specified in advance. Davis et al. (2011a) have compared the use of Logic Programming to a traditional parametric model schema for a simple design task and found a five-time increase in speed in order to create the model. Although Davis does sound word of caution that it is hard to generalise from such small studies (2013, p.71), it does however still show a promising direction for improving

parametric flexibility even if an (undirected) associative network must still be specified in advance.

Strategies for implementing bi-directional constraint-based design exploration were proposed by Kilian (2006), and more recently developed by Coenders (2012) for his 'NetworkedDesign' framework. These approaches give more flexibility in changing constraints to objectives and vice-versa, but still rely on predominantly top-down manipulation by the designer.

4.6. Discussion

As we have seen, there exist methods that offer improvements to both the speed of graph manipulation and their legibility as collaborative models. However, most of these approaches are all still based around wilful top-down modifications being made to a single graph topology that must be explicit and legible. In this sense, they do not address the problems found on the Escher Tower project where potentially millions of different parametric schema are required that cannot be generated manually. In summary, parametric design seems unsuited to healthy design exploration because the design team must pick a limited number of building typologies, or else engage in the laborious task of manually creating different parametric definitions for each - an unrealistic task at the conceptual design stage. Davis' studies in Logic programming suggest an alternative approach, but focussing on axioms and allowing the machine to assist in constructing the relationships.

4.6.1. Relation to Software Development

In software development, two conflicting views of how to manage a project are popular. The first, known as the 'Waterfall Model' focusses on a linear mode of development, investing time up front to understand the problem before any design development takes place. In relation to building design, this would see the design team fully understand the brief before developing and then proposing a single design to the client. The second is known as the 'Spiral Model' of development (Boehm, 1988), which acknowledges that understanding requirements and project goals evolve during the design process itself, and hence the design team must be *agile* in how they approach a problem. In building design at the concept stage we can see parallels with the latter, as architectural problems are wicked in nature and require us to reassess project goals and constraints iteratively. As Lawson (2006, p.56) states: "design

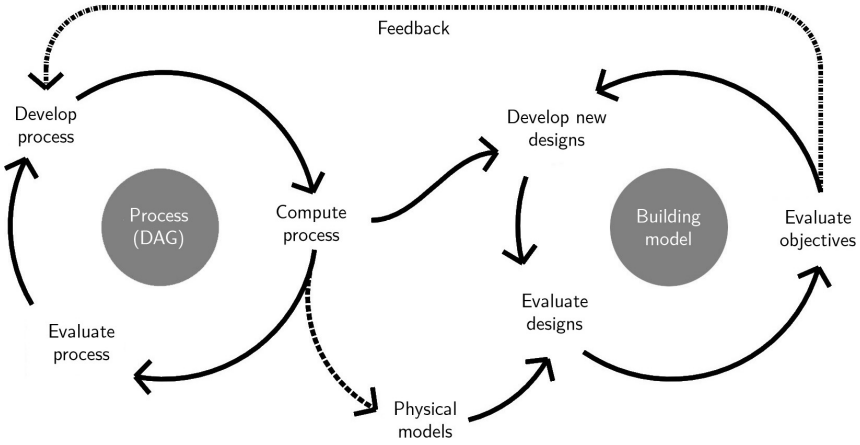


Figure 4.8: A building model develops alongside its computational generating process

problems are often both multi-dimensional and highly interactive". We can therefore say that unless our computational process is agile, it will not be well suited to the complex task of building design.

As the building is the result of the algorithm, so modifications to the algorithm will create modifications to the building (fig. 4.8). With parametric models, both are visible to the design team. A computational approach must also work alongside other modes of representation and processes, such as the creation of physical models. Although the latter is possible especially through developments in rapid prototyping, a limited number of options can be feasibly realised using this method. In practice, a combination of both computer and physical models helps understanding from different stakeholder perspectives (Hanna & Turner, 2006).

As opposed to computational method that imposes a known heuristic, here there is no such well defined approach from the outset. Instead the approach must work as a structural coupling (Maturana & Varela, 1980) between humans and computers which influence each other in a conversation during design development. Front ending design intent as per the Waterfall method does not resolve the issue that earlier in the project there is the more the potential for design constraints and objectives to change. Indeed, it may be that specific design objectives are left open for the building users to explore and interact with, as explored by architect Cedric Price (Mathews, 2007).

4.6.2. Review of Aims and Objectives

In this Chapter, I have explored the use of parametric modelling at the early stages of design. Through looking at case study projects both myself and in literature, it is clear that there exists a conflict between the inflexibility of the parametric model created top-down and the wicked

problem of early stage of design. On a positive note however, shifting to workflow that enabled constraints and objectives to change over time was working successfully on the Escher Tower case study project.

In response to these findings, the exploration of computational modelling methods that avoid fixed body-plan are explored in the next Chapter. This involves moving away from explicit parametric models and towards implicit complex development processes in generating form.

5. Freeing the Body Plan

> "Technology is the answer, but what was the question?"
>
> Cedric Price, 1966

In the previous chapter, I argued that the inflexibility of the modelling process becomes a significant stumbling block when attempting to explore a wide range of forms at the concept design stage. To this end, the promise of parametric models to offer better design exploration was in reality not the case because too much time was spent constructing the associations that then constrain the design space too early. Instead, I argued that the generative algorithm itself should be more adaptive rather than just the parameters that control a model constructed with assumed intent. On Escher Tower at least, we needed to become more agile with the programs that generated the form during design exploration.

Computational approaches that involve a flexible body-plan are therefore more suitable than parametric models at the early stage. As we will see, these approaches often incorporate implicit aspects that are not possible with a DAG structure used by parametric models. By negating such a fixed algorithm/structure in the generation of form, emergent processes that adopt an implicit and not an explicit approach may offer a suitable alternative where complexity and emergent architectural form are able to develop (Frazer, 1995). Here there is a direct analogy to biology in the development of form.

5.1. Introduction

Unlike parametric models, the study of development in nature from an embryo to organism (embryogenesis) leaves no easily understandable trace of how to build an organism. Instead, the DNA of a creature contains rule based information that encodes an emergent process of development of form from a single embryo, i.e. morphogenesis. The genotype is the base level representation of the organism (for example

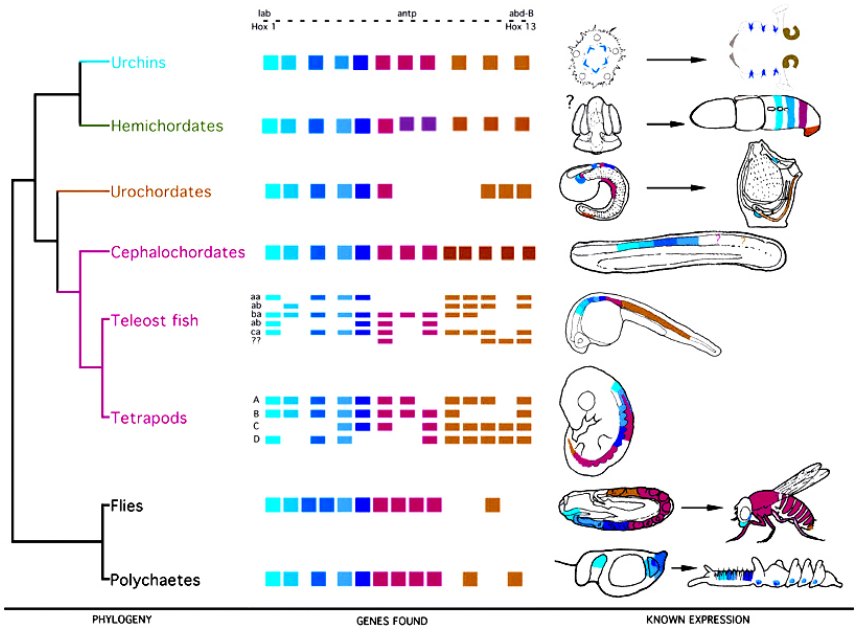


Figure 5.1: The relatively small set of hox genes play a significant role in setting out the body-plan of an embryo for different creatures

DNA), with the phenotype being the final form following development. The process from genotype to phenotype is *implicit*, with complex order emerging from low level rules (Turing, 1952). Due to this complexity, the process of building a living organism from its genotype even for the simplest of creatures is only just beginning to be fully understood.

5.1.1. Evolutionary Development

The development process from genotype to phenotype is combined with natural selection in order to evolve organisms capable of surviving in their environment. This combination between individual development (ontogeny) and the evolution of species and lineages (phylogeny) is known as Evolutionary Development, or Evo-Devo for short. As a complex adaptive system, natural systems are able to adapt the design of living creatures to suit an ever-changing environment in a coupled relationship (Carroll, 2005).

The complexity of an emergent process means that unlike with an explicit one, small alterations to the genotype can lead to large changes to the phenotype, for example its size, shape and number of repeating modules in the living form (Weinstock, 2010). There exists a specific part of the genome known as the Homeobox genes, a relatively small part of the entire genotype, but in combination with morphogens play a significant role in setting out the body-plan of the organism (fig. 5.1) (Wolpert, 1991). This process regulates the growth rate of new cells, with timing changes to these rates resulting in large phenotypic differences (Gould, 1977).

5.2. Artificial Embryogenesis

The ability of natural systems to produce a great variety of 'designs' has offered inspiration for computational designers involved at the conceptual stage (Steadman, 2008). Artificial Embryogenesis (AE) is the study of taking natural evolutionary development and artificially replicating it inside the machine to generate new designs automatically. In short, AE consists of the use of a non-linear system to map from representation to organism, inspired by the mapping from genotype to phenotype in nature (Kowaliw, 2007).

Bentley & Kumar (1999) used the concept of Artificial Embryogenesis in order to compare various ways of generating designs and assess how appropriate they were for the task of wide exploration in combination with an evolutionary algorithm. They found that an implicit representation gave rise to a *much greater variety of designs* than an external embryogeny such as a fixed parametric model with only linear metric parameter variation. With an implicit approach, small changes in the genotype could give rise to large changes to the phenotype resulting in a wider exploration of the problem domain.

Various studies in using an implicit embryogeny have been conducted that aim to relax the body-plan by losing an explicit representation of the generation process altogether. This was first explored by Bentley & Wakefield (1998), Coates *et al.* (1996) and Frazer (1995) in the evolution of rule based cellular automata (CA) that generate form. Here, the genotype encodes the simple rules from which the CA simulates deterministically to produce the phenotype. Multiple genotypes can then be manipulated as part of a metaheuristic search by assessing the phenotypes and selecting good solutions either artificially (Dawkins, 1986) or automatically (Goldberg & Holland, 1988). As opposed to defining the body-type explicitly, the idea behind such methods is to find a neutral description of 3D form which is capable of embodying the widest possible range of objects. Local rules can also embed design team heuristics as well as the global assessment of the final form.

5.2.1. Local Rules

Due to a deliberate loss of control in using an emergent system, the traditional role of architect as top-down 'master designer' is to some extent replaced with bottom-up 'systems designer' (Kalay, 2004). In reality, the designer or design team is still in charge of setting up a process and assessing results, and hence must also assume some form of control. However, by acting at this meta-level of abstraction, the machine is

now in a position to suggest novel and perhaps well-performing designs that may have been beyond the reach of the human with already pre-conceived ideas. A bottom-up approach to design also changes the emphasis to think locally about the rules that generate form, as opposed to globally with a top-down system. As Coates (2010, p.167) states:

> "The use of computers to explore emergent form has one advantage that, in order to build the morphological system, you have to define it in a formal language which brings the structure and meaning of the model proposed into the realms of expression and debate."

The next section explains a case study project whereby local rules and global outcomes are both influenced by the stakeholders and open for change throughout the design process. In response to the inflexibility of parametric models in the exploration of form, an emergent approach with no fixed topology was then implemented with participant observations recorded.

5.3. Implementation

Following the collaboration with Bjarke Ingels Group on the Escher Tower Project and the subsequent findings, I was invited to work on a further project at the concept design stage: the ENI HQ Competition [20Eni] in Milan, Italy. The project was for a new commercial office on an industrial park for which BIG wanted to reconsider the idea of a standard office block typology to something that would also involve public space.

Their initial geometric concepts were based on the formation of graphs representating spatial layouts (fig. 5.2), in reference to the existing ENI owned buildings on the site which created some context to the proposed intervention. These graph structures gave an abstract representation of the spatial structure of the office and became a way of describing different geometries using a different mode of representation. Although spatial phenomena do not stop at some clearly defined geometric threshold such as representations like graphs (Derix & Izaki, 2013), the specific nature of this project made them an appropriate choice. Plan graphs of spatial adjacency were first used by Steadman (1983) and later Jupp & Gero (2003) and remain a useful method of generating and analysing spatial aspects of design (Hillier, 2007).

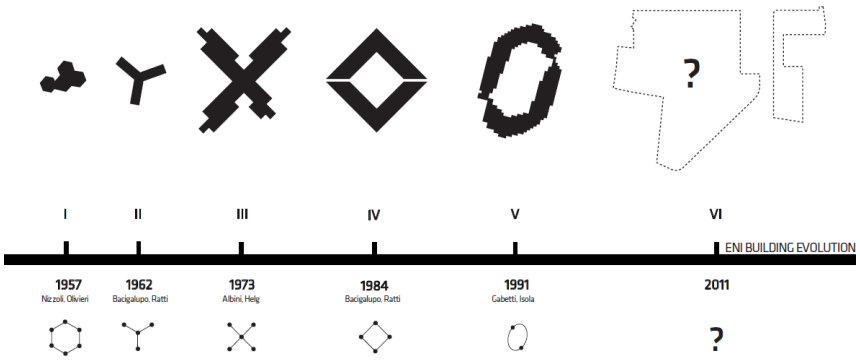


Figure 5.2: Graph representation of local buildings (Image courtesy of BIG architects)

5.3.1. Modelling

As with Escher Tower, the architect's initial foam models (fig. 5.3) provided a visual guide in terms of the variety of forms to be produced in the machine. The modelling process developed here is an example of *procedural modelling* whereby a set of simple rules generate more complex forms. The graph representation provided an abstract concept for the formal development because although not being completely unrestrained, it was still open to manipulations in body-plan. The project therefore provided a good chance to escape the problem of parametric modelling software at the concept design stage as discussed in Section 4.5.1.

Graph structures are relatively easy to manipulate geometrically and topologically and therefore could produce a wide range of morphologies. A similar approach to generating spatial structures with alternative topologies were pioneered at the urban scale by Hillier and Hanson (1984) for investigating "de-geometrised" village layouts and Coates (2010, p.153) with his similar Alpha Syntax model by establishing a formal shape grammar with simple geometric and topological rules governing the growth of possible designs.

5.3.2. Local Rules

The rules used to generate various forms for the ENI HQ design were based both on the molecule analogy from BIG and the requirement to generate similar structures to those already made using foam models. For example, local rules which generated angular forms similar to the models were embedded in the graph structure generator.

An agent based model was used in order to control the node location geometry using an attract/repel force similar to the self-organising process used on Astana National Library [01Anl]. A stochastic shape grammar was used to change the topology of the molecules similar to

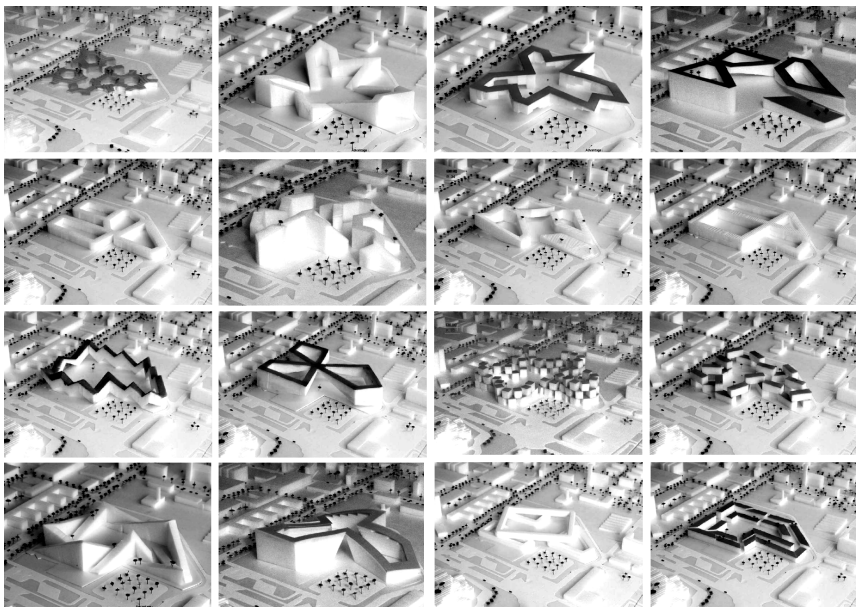


Figure 5.3: Initial foam model concepts (Image courtesy of Bjarke Ingels Group)

stochastic Lindenmayer Systems (Prusinkiewicz & Lindenmayer, 1990, p.28). Nodes are able to attach and detach neighbours, as well as grow new edges during the development of form. The generative process therefore had seven parameters that control the development of form over time:

1. Growth probability (0 - 0.05). Controls the probability at each iteration for the graph to grow an extra edge and randomly located an extra node.

2. Attach neighbour probability (0 - 0.1). Controls the probability at each iteration for the graph to add an edge to its closest unconnected neighbour.

3. Detach neighbour probability (0 - 0.1). Controls the probability at each iteration for the graph to remove an edge to its closest connected neighbour.

4. Flesh width (0 - 14m). Adjusts the width of the buildings. An assumption is made that every building has the same office floor width.

5. Node height factor (0 - 40m). Controls the extrusion of each node from the ground plane to form the 3d buildings.

6. Attraction/Repulsion strength (-0.05 - 0.05). Neighbouring nodes influence each other with a push or pull force given by this parameter.

7. System damping (0.95). A second order multiplier that damps parameters 1, 2, 3 & 6 towards zero.

For the seven parameters, the range of values (bounds) were set manually to give a range of outcomes similar to the physical models. They also

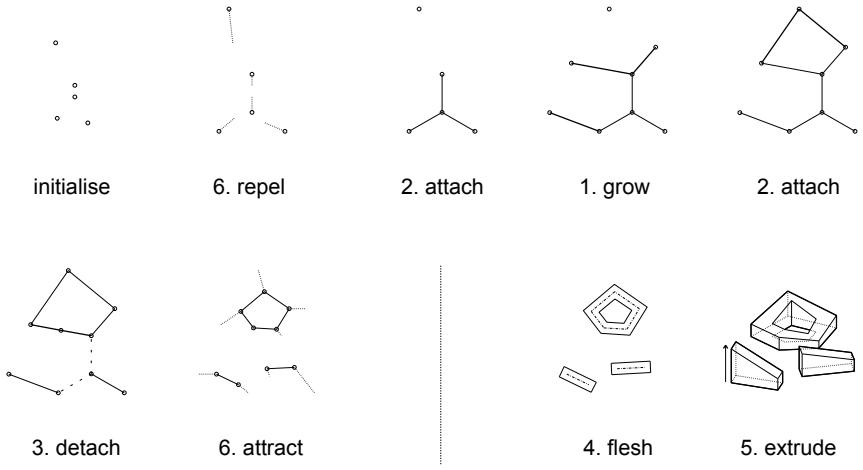


Figure 5.4: Development of form using the ENI graph generator

embedded some architectural constraints, such as the maximum width of the floor plate (for passive ventilation).

The reason to use a nearest neighbour approach to alter the topology of the graph was partly due to planarity. Nearest Neighbour Graphs (NNG) are known to be sub-graphs of Delaunay triangulation and therefore maintain planarity (Eppstein *et al.*, 1997), meaning that buildings would not usually intersect each other (this is not guaranteed, only due to the continual movement of the nodes). This planarity property has been useful in generating spatial layouts on the plane for other architectural applications (Harding & Derix, 2011). The graph generator was developed as a Java application and could be used manually or automatically.

5.3.3. Performance Objectives

As with the Escher Tower Project, there were certain ideas that the architect already had in mind that influenced the performance objectives decided upon by the design team. For BIG, this didn't simply mean the local building geometry, but also the historical use of the site, leading to an intent to have public gardens on the roofs of the buildings. Other factors such as the site context and the local climate influenced the need to reduce solar exposure and hence solar gain in the summer months for the offices. The main initial objectives were as follows:

1. Minimise solar exposure. This was measured as an accumulative value over the summer months by reusing the analysis algorithm I had previously developed for the Maljevik Bay Project [19Mjb].

2. Maximise green roof area. This was seen as an important design driver for BIG in attempting to change the concept of how a private office building was used. Here, the public were permitted to walk

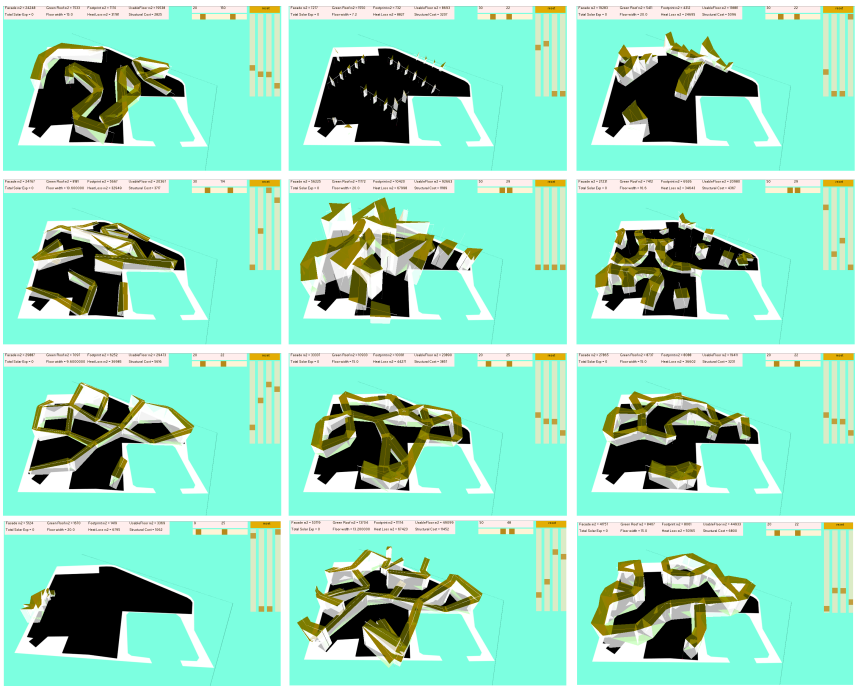


Figure 5.5: Screen-shots from the Java application

upon the building roof and hence maximising the amount of roof area became an objective.

3. Minimise total heat loss. A good rule of thumb for building heat loss is that it is proportional to its surface area Heywood (2012, p.76).

5.3.4. User Control

The application was initially developed with manual controls, with parameters adjusted and the growth process taking place until fully damped (fig. 5.5). The design team was able to grow building forms on the site and observe the status of the performance objectives in real-time. However, when using the software it was difficult for the user to understand how to control the results. Even by enabling a random seed for the growth probabilities (thus making the algorithm deterministic), the generator was found to have a high degree of epistasis, or in other words, slightly different initial conditions led to completely different results. Although this enabled a wide variety of designs to be generated, there existed no strong connection between what adjusting parameters was actually going to achieve in terms of the final design. The complex process left the user somewhat distanced from the process, a finding discussed further in Section 5.4.

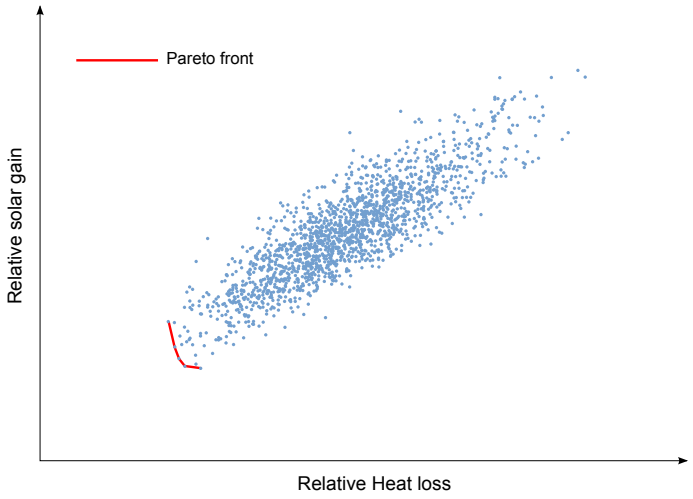


Figure 5.6: Brute-force search of outputs with two objectives

5.3.5. Using a Metaheuristic

By setting the performance objectives outside of the main algorithm, the task of generating designs could also be attached to a metaheuristic algorithm to give the design team a *good* selection of designs. A simple brute-force approach was used to uncover a good set of solutions by sampling 3,000 different final design options explored within the parameter limits. The results are shown in fig. 5.6, with two objectives compared. A sample of design options close to the Pareto front could then be selected and taken forward by the design team by recording the initial condition (random seed) and the parameters that generated the final result. Indeed, although not known beforehand, the results actually showed a strong correlation exists between solar gain and heat loss, meaning that they could perhaps be combined into a single objective.

As previously discussed, the system was highly sensitive to small parameter adjustment especially with regards the topological controls, meaning two similar designs were often generated using vastly different parameters. This nature made it difficult to make any wilful impact on the development process after the final network had been generated. Although CAD files could be exported of good geometries for further work, the graph generator application itself felt like a black box, distanced from the design team.

5.4. Observations

Although the algorithm generated results similar to the foam models, in the end BIG did not end up using the software application *at all*, as either a manual application or as a generator of good designs. This was disappointing, and can be seen as a failure of the approach, but it

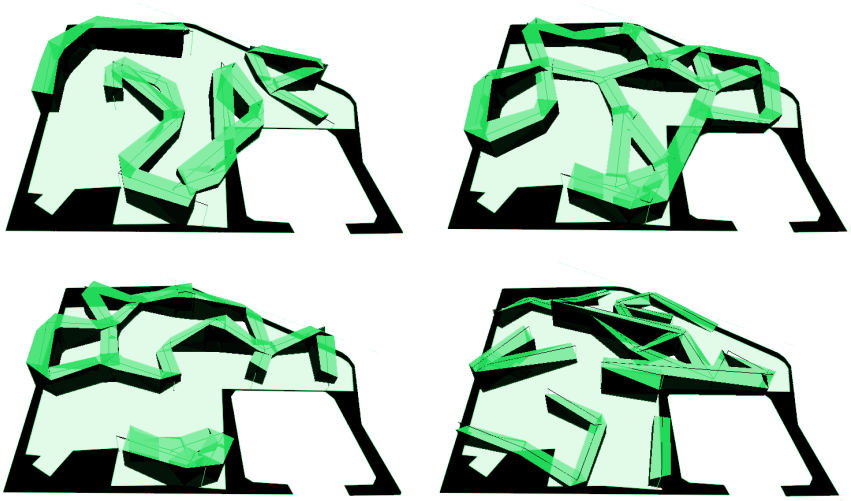


Figure 5.7: A sample of good designs close to the Pareto front

highlighted a general problem with generating software applications that generate design models where there is no engagement in the process by some stakeholders. *Control* of the model becomes an important issue, and although computational methods can support design, one must feel connected to any process involving a machine. This is discussed further in Section 5.4.2.

Understanding process and feeling involved is human nature. In 1997, the chess champion Gary Kasparov lost a six game match against IBM's Deep-Blue, the first time a computer had defeated the best that humanity could offer. Following the loss, Kasparov demanded to see the computer logs to understand how the computer was thinking when a crucial match changing move was played. Upon being denied the information, Kasparov concluded (incorrectly) that the move had in fact been made by a human to gain a territorial advantage. As well as its interesting relation to the Turing Test, it also shows how false conclusions can occur if process is not made explicit. The same can be said of emergent design methods. Whilst it is true that such approaches help to quickly explore vastly different designs and investigate novel design possibilities, an important question is often overlooked. If the design team are in control of setting up an implicit process and able to cognise the final results, what level of intervention is possible *during* the design process itself if complex processes are effectively a black box of irreducible complexity?

In Kumar's AE example discussed in Section 5.2, whilst the implicit approach gave wide design exploration, only the final resulting phenotype was then available with no simple description of the growth process (i.e. genotype to phenotype) that could be understood by the designer. This is exactly the same problem science has when attempting to understand for example the complex system of embryogenesis. In effect, the entire (evolved) simulation has to be run again from scratch and followed step

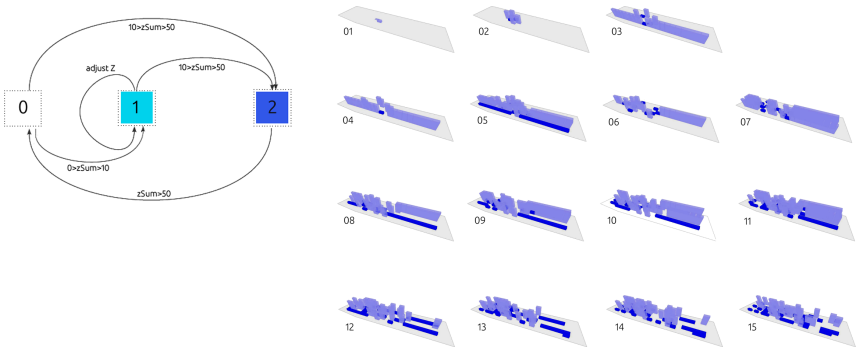


Figure 5.8: Massing generation using a cellular automaton with simple low level state transition rules

by step, and even then it is hard to see how the human can step in and affect such a complex process because any intervention can still lead to vastly different outcomes.

This is the case even with deterministic systems that display chaos, such as the three-body problem (Poincaré, 1914). Any intervention at any stage of the process (not just the initial conditions) can cause vastly different outcomes at a later time, similar to the ENI graph generator. This sensitivity is also found in artificial complex systems such as certain class IV cellular automata (CA), for example Wolfram's Rule 110 (1984) or Conway's game of life (Gardner, 1970) which are computationally irreducible.

That is not to say complex processes should not be of fundamental importance in computational design; far from it. Indeed, the massing options quickly generated on the Pluit City project [32Plu] utilised a complex approach to generate a wide range of results from very simple density rules (fig. 5.8). One must however understand the implications on collaborative practice, and why *sometimes* a more explicit description of process can be beneficial.

5.4.1. Collaboration Issues

Why does an explanation matter? Although emergent processes are certainly interesting and do have architectural applications, their appropriateness as part of a collaborative process is questionable, because the generation process behind the form must to some extent be made explicit so that we may interact with it, and not just initial conditions and/or the final result. If humans and machines are to be used in a conversation, then a language in which they communicate has to be understandable by both, even if each will have a different interpretation, or form different mental models. In essence, the computer must enable the human to understand *how* it forms a particular result if it to be seen as a stakeholder in the design process and not simply a slave.

One only has to look at automatic plan generation (Jo & Gero, 1998; Gero & Kazakov, 1998; Eastman, 1973; Grason, 1970) to see why this is important. Although many papers have been written on this subject and many efficient algorithms developed in isolation, hardly any applications of such software have been implemented on real projects. As Liggett (2000, p.213) concludes in her survey about layout automation at the turn of the century:

> "In spite of the long research history associated with automated layout and space allocation systems, in practice these systems have not been utilized to their full potential."

I would argue this is because design is a process of understanding *wicked* problems, not something that is easily generalisable and hence wrapped up into a tool. Black boxes for other designers to simply deploy ignores the essential issue of control and inclusion; as Derix states (2010, p.61):

> "technological ambition side-lines the designer into a seed watch evaluate role who feels his/her intentions and heuristics are not participating in the search."

The software developer is the one attempting to emulate a typical human designer, instead of setting up a system to enable him/her to engage in conversation with a machine. This application of externally imposed design management strategies from academia is still just as prevalent even today (Ren *et al.*, 2011).

Instead of developing software to automate the *logic of architecture* (Mitchell, 1990) in a systematic general approach (Pahl *et al.*, 2007), I would argue that we need to look at communicating with machines not using them to implement overriding frameworks that dictate the design process. To this end, Building Information Models (BIM) *have* shown the way for better understanding of geometry by allowing different semantic data from different stakeholders, but they also make assumptions on what geometry should be, for example by having ready made tools for making 'walls' and 'roofs.' Whilst this is good for the standardisation of building component, it is too specific for healthy conceptual design (Cavieres *et al.*, 2011).

Parametric models offer a way to engage at the level of process explicitly, so long as they are flexible enough and legible to different stakeholders (see Section 4.2.2), a point made by Coates (2010, p.167):

> "The use of computers to explore emergent form has one advantage in that, in order to build the morphological system, you have to define it in a formal language (define an algorithm) which brings the structure and the meaning of

> the model proposed into the the realms of expression and debate".

Effective communication between stakeholders during design development requires understanding, and this comes at the cost of speed. In machine learning in recent years, humans have begun to give up on understanding how machines solve problems such as language translation, instead relying on complex statistical pattern matching processes. With such systems, an explanation of how something has been solved is not important, as long as the program works it doesn't matter how it did it (Russell *et al.*, 1995). However, we have seen in architectural design that it *is* important if humans are to be engaged in the process, because of the equally complex nature of design itself and the importance of explaining to others how (and sometimes why) something has been done in a collaborative environment. A similar medical analogy given by Heaven (2013, p.34):

> "What if a machine learning system decides that you will start drinking heavily in a few year's time? Would doctors be justified in withholding a transplant? It would be hard to argue your case if no one knew how the conclusion was arrived at."

Understanding *how* in this medical example may lead to a suitable treatment in the future. We have seen in Section 4.5.3 that with parametric models it is the component functions and graph topology that guides future variation. The result maybe the same form, but how it was arrived at will govern any future development. As a final point, one may note the similarity here to convergent evolution in natural systems - for example the fact that vertebrates and octopuses developed the camera eye completely independently. Or for example, the development of placental and marsupial forms with completely different evolutionary strands as described by Dawkins (1986). Understanding evolutionary development gives a different categorisation for organisms, completely different to that made by simply looking at them and recording features.

5.4.2. Case Study Evidence

Learned experience of individual stakeholders forms the main driver behind the design process, and hence this must work alongside any computational process not embedded in the algorithm itself using a set of 'human instructions'. In an semi-structured interview conducted with BIG employee Catherine Huang following the ENI project (21st November 2011), she commented that [using the software described in Section 5.3] they could not access the process itself and therefore the

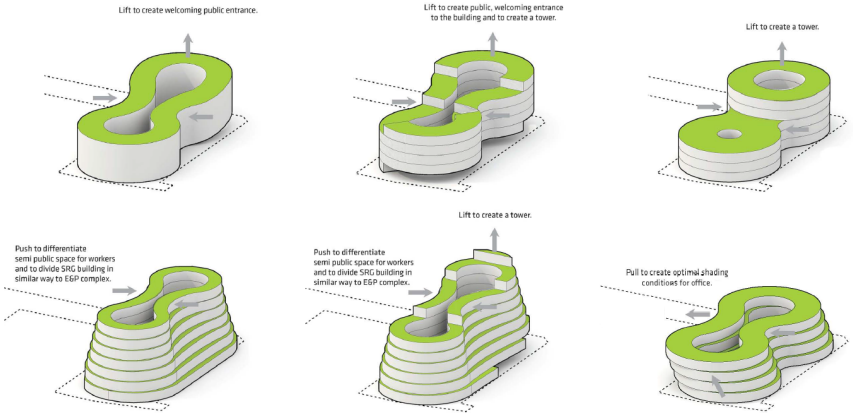


Figure 5.9: Geometric operations on a later design for ENI based on human experience (Image courtesy of BIG architects)

idea of other stakeholders writing software that conducted aspects of the design for them without knowing how it was done was troubling. In collaborative design, understand *how* leads to discussion as to *why*. These may well be hard to define, involving tacit knowledge or otherwise but it allows the stakeholders to at least consider the question. In essence, any geometric process that gives rise to a building form being made explicit can benefit collaboration, just as with the resulting building model itself, because multiple stakeholders may view the reasons behind it in different ways.

Huang stated that this explicit record of process was important to how BIG explain the reasons behind the development of form in project reports issued to clients and other consultants. Such "geometric gestures" were part of a series of processes learned from previous projects and had become a language of design *specific* to the architectural practice. This was assisted by the saving of hundreds of foam models of old projects, including designs that were not progressed. These acted as reminders of the meaning behind the process for them. However, it would be wrong to assume this language was fixed, indeed Huang commented that such rules of thumb were in a constant state of flux and although they assisted learning from their previous work, they were open to new directions as to avoid becoming a fixed design system.

These gestures take on different meanings for different stakeholders. For example, creating cantilevers at each level my help with shading (fig. 5.9), however the structural engineer will immediately view this gesture differently. The fact that BIG communicate how and why to other consultants opens up the debate, as different stakeholders will have different world views of a geometric process as well as the final resulting form. Note how this differs from Sections 2 & 3 of this thesis, whereby a single heuristic was embedded in the generation of form as opposed to many (see discussion, Section 3.4.1).

5.4.3. Evidence from Industry

Before the advent of parametric models, Peter Eisenman was one of the first architects engage in generative process that could be recorded as a cognitive artifact in the form of diagrams, able to be interpreted differently by others. As he states (Eisenman & Somol, 1999, p.169), "diagrams became a means to uncover something outside of my own authorial prejudices". In a series of projects including House VI, the generative *traces* or *diagrams* of process are expressed in an explicit form in the final construction itself. This purely geometric process *is* the building and leaves itself open to interpretation by everyone, although the level of collaborative engagement with a completed building is by that time non-existent.

In contrast to Eisenman's desconstructivist approach, when geometric operations have associated meaning behind them *during* the design process they are often governed by particular (human) motivations. These are commonly known informally as rules of thumb or "guiding principles" (Lawson, 2006). For example, architect Ken Yeang developed and subsequently published a set of geometric principles for the design of environmentally responsible towers after years of experience in the field. Due to the measurable nature of the performance goals, the success of such gestures could be isolated and measured, and therefore generalised to some extent and used by other design firms. Likewise, the architect Carlo Scarpa used to align geometric operations with *how* an object would be sculpted by a craftsmen. Burry (2013, p.103) recalls how in describing Gaudi's process of designing a column, the computer script becomes important in telling the story to others: "Scripting the narrative seems crucial if we are to animate the sequence and afford more general intelligibility."

Parametric models with an explicit DAG representation allow stakeholders to form their own mental models of design process (Checkland, 2000). Again, this sits in contrast to Building Information Models that allow stakeholders to work together only at the final building model level and associated semantics (Succar, 2009). A word of caution should be sounded however, in that parametric design software (such as Grasshopper) currently has a strong bias to only geometric processes. Aspects such as the placement of lighting or the choice of colour and material for example will also affect the experience of space. In fairness, such features are beginning to be made available to parametric models, however a parametric model will rarely encapulate *all* aspects of a design and must work alongside other phenomenological aspects that may not be explicit in the program.

5.5. Discussion

The crucial learning from this chapter was that BIG make explicit their reasons for geometric gestures in order to explain how and why a particular form comes into being. This "cognitive artifact" of process helps designers learn from previous work as well as helping explain *how and sometimes why* to other stakeholders. To obscure this using an implicit process caused unforeseen problems on the ENI project. Stakeholders were unable to understand how the machine had arrived at a particular design and hence could not engage, instead feeling that the machine had imposed a particular approach against their will.

As Frazer warned (1995, p.18), "If used unimaginatively, computers distort criticism to the end product rather than to an examination of process". By obscuring the geometric process as with an irreducible complex system, we are destined to focus only on the initial rules and subsequent final outcomes and obscure the in-between. As the author Douglas Adams writes (2002), "If you try and take a cat apart to see how it works, the first thing you have on your hands is a non-working cat". Design is not attempting to *replicate* how cats are formed or how cat consciousness works - that is the domain of science. Design is about understanding to create new meaningful interventions. Here we must begin to question whether the black box of self-organisation as a collaborative design approach is compatible with humans that often crave involvement and control.

By acknowledging this reality, it becomes clear that we should seek a middle ground that offers the flexibility of an implicit approach with the cognisability and common language offered by an explicit one. Perhaps instead we need to remember the benefits of parametric models in making process explicit, whilst somehow increasing their flexibility? In response to both the problems encountered when humans attempt top-down control of the explicit parametric graph, and the irreducibility of a complex processes, might it be possible then to think topologically and automate the generation of the graph itself? This would opening up exploration of different building typologies as required at the concept design stage whilst still maintaining an explicit representation of development from genotype to phenotype (fig. 5.10). Such a process would be analogous to a creating an explicit embryogeny in describing the process of form-generation, sufficiently agile to adapt to the changing nature of the design process.

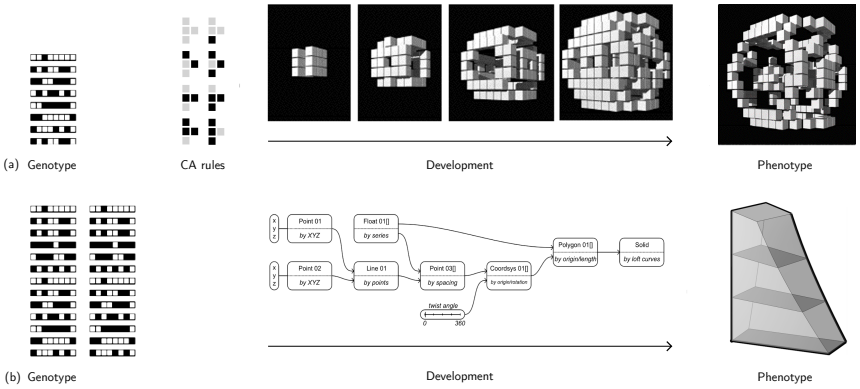


Figure 5.10: Comparison between an implicit (a) and explicit (b) embryogeny process for generating simple forms. CA example taken from Coates et al. (1996)

5.5.1. Reconsidering Parametric Models

A parametric model representation is often complicated but not complex, and therefore theoretically knowable (Kurtz & Snowden, 2003). Form generation from simple components is ordered in a hierarchical way. Although a complex implicit approach gives more freedom, the complicated explicit one gives better understanding. As Aish comments, (pers. comm., 8th November 2012):

> "Early code generators applications created code that 'worked' but code that was verbose and unfathomable by 'human programmers'. The 'holy grail' for those of us developing a code generation system was to create code generators that in turn create code that was human readable.. i.e. understandable."

As we have seen in Section 4.2.1, as well as the final form, an additional parametric representation of *how* the form is made does indeed have benefits in a collaborative environment and therefore to some extent helps negate the problems that black-box processes such as automatic layout planning run into in terms of adoption by others. Successful collaboration has been made from the early stage by using parametric models, however these have been limited to a single building typology that has been defined from the outset - for example the stadium 'bowl' typology (Hudson *et al.*, 2011).

Certain standards must be made when generating a model that was understandable as a collaborative artefact and not a tangle of spaghetti (Davis *et al.*, 2011c). By considering the machine as an additional stakeholder, if parametric models are automatically generated we should make sure they are 'clean' and understandable so that interaction can take place.

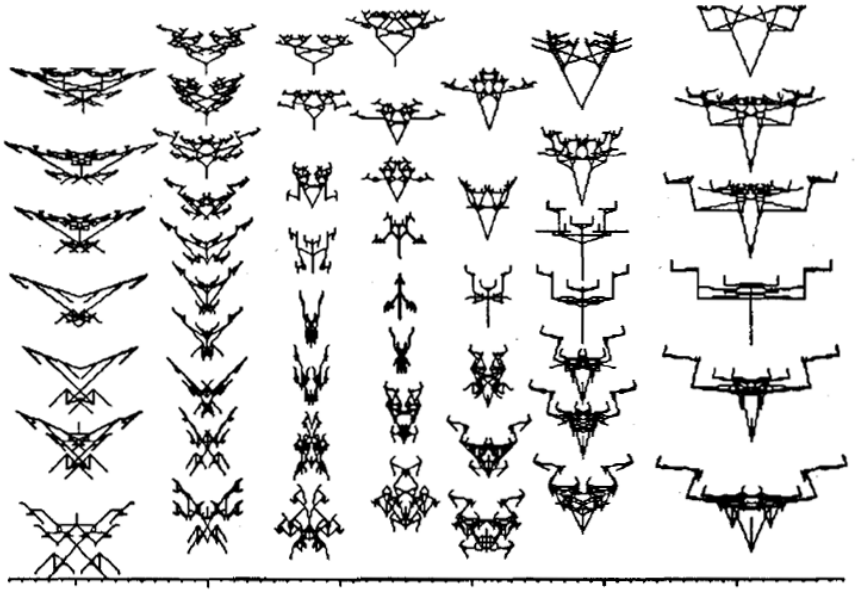


Figure 5.11: Examples of biomorphs. Source: (Dawkins, 1986, p.70)

5.5.2. Evolvability

In Section 4.5.2, it was discussed that Dawkin's Biomorphs and Sims' creatures could evolve a wide variety of forms even if their genotype to phenotype mapping was explicit, because their topological structure was free to change during a search process. However, another property known as 'evolvability' was also present due to this mapping. fig. 5.11 shows some Biomorph designs taken from The Blind Watchmaker (Dawkins, 1986) laid out from left to right according to their similarity in 9-dimensional genotype space (something Dawkins calls a "genetic ruler").

By visual inspection, we can also see that such a similarity is also present in phenotype. Dawkins later remarks in his book The Evolution of Evolvability (Dawkins, 2003), that the creatures were so easy to evolve because they had this direct mapping between genotype and phenotype. The closer relationship between the genotype and phenotype than with an implicit approach leads to faster adaptation. This mapping does however come at the cost of efficiency. Within biological organisms, we saw in Section 5.1.1 that a small mutation in a single homeobox gene can cause a large change in the body plan. With an explicit system however, a larger genotype is required because no instruction can emerge autonomously with an explicit embryogeny (Senatore, 2009).

Kumar & Bentley (2000) have also found that achieving evolvability with an implicit embryogeny is more difficult than for explicit one because of the discontinuity it causes in the search space, i.e. a high degree of epistasis. Epistasis is the degree of dependency *between* the genes in a genome.

This distinction between explicit and implicit is also known in the Artificial Embryogeny community is known as the Cell Chemistry Approach vs Grammatical Approach to encoding (Stanley & Miikkulainen, 2003), and in neural networks as the difference between non-modular and modular designs. If we look to nature, we may think that the emergent process of development from genotype to phenotype is completely implicit, however it has recently been found that a key driver of evolvability in natural systems is the widespread modularity of biological networks - their organization as functional, sparsely connected subunits (Clune *et al.*, 2013). In reality then, a mix of explicit *and* implicit embryogenies is probably how natural systems can have such complexity whilst still remaining highly evolvable.

Although an explicit parametric DAG is not technically *complex* but instead *complicated*, if its structural development is combined with a metaheuristic search such as a genetic algorithm, complexity lies not at the level of generating geometry, but in what happens during design exploration, i.e. in the generation of good explicit embryogenies quickly and efficiently. This suggests a direct mapping between genotype and phenotype (low epistasis) makes such a process easier to follow and engage with. As Derix states (2010, p.65):

> "...the designer can better interfere with computational heuristics and understand the search struggle, offering the opportunity for identification between designers' analogue and computational heuristics, thus enabling the validation for wicked problems when no explicit goals are set."

5.5.3. Review of Aims and Objectives

In Chapter 2 it was argued that our current role as engineers had to move to the front end of the process in order to be effective in computational design. In response, Chapters 3, 4 and 5 documented several different approaches to early stage collaborative design using alternative computational methodologies that influence the workflow between stakeholders. The key learning points from each chapter are as follows:

- We should *not embed a particular heuristic too early* and therefore impose an overriding idea before the design problem is known (learning from Chapter 3)
- We should *allow large body-plan variations* for healthy design exploration of different building typologies (learning from Chapter 4)

- We should *provide an explicit representation* of the geometric process for better stakeholder collaboration (learning from Chapter 5)

If we cannot achieve the last of these, the key question becomes do we attempt to make the complex approach more understandable, or the complicated approach more flexible? For this research, I decided to take the latter approach due to the known benefits of existing parametric modelling software in a collaborative environment. However, this decision was a conscious choice, acknowledging that this may come at a loss to the amount of complexity an implicit system brings.

The next chapter explores methods in making the parametric model more flexible by investigating ways to alter the graph functions and topology in addition to metric parameters. Here I turn to Genetic Programming, the study of generating programs with programs. In this context, this is development of programs that generate *visual* programs.

Part IV.

Bridging The Gap

6. Meta-Parametric Design

> "There is one possibility that has always intrigued the lazy coders, what if we could get the computer to write the code for us - so we can just write one program that evolves a program, and then we can all go back to sleep!"
>
> Paul Coates (2010)

6.1. Introduction

We have seen in Section 4.2.1 that parametric design software allows a visual representation of a geometric process, formulated during design development. It was argued that whilst the creation of the parametric graph structures remains predominantly under top-down control, the vast search space at conceptual design stage cannot be properly explored due to topological inflexibility. However as shown in Section 4.2.2, the explicit nature of parametric models does have collaborative benefits. In response, by thinking at a higher level of abstraction, the generation of the parametric graph structure itself can be *agile* alongside modifications to the metric parameters (Harding *et al.*, 2013).

Such an approach takes inspiration from Genetic Programming (GP) whereby the automatic generation of computer programs takes place as opposed to simply adjusting variables. Automatic generation of parametric graph structures would enable different building typologies to be explored, even if the variation of the graph structures was fairly minimal. It is proposed that such a system could either be controlled manually, be incorporated into a feedback loop using a metaheuristic solver or indeed include aspects of both.

The crucial aspect to emphasise is that such a system is not proposed to *replace* a design team, but instead enhance its capabilities through computation by approaching design problems at various levels of abstraction. Far from 'all going back to sleep' as the lazy coder might dream, we still need to be very much awake... if not more so!

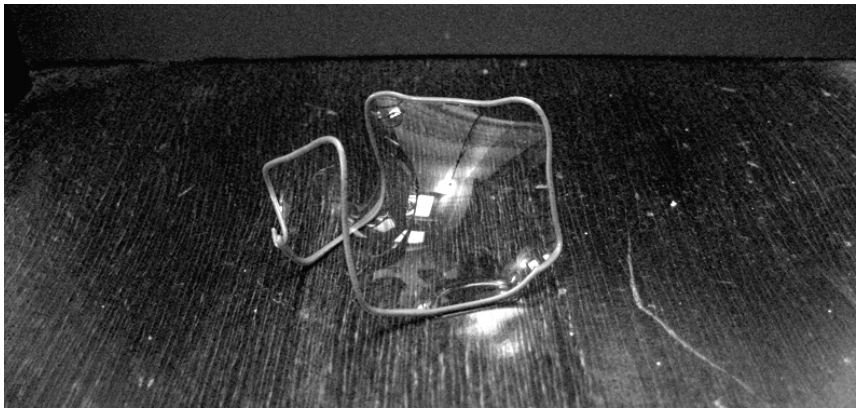


Figure 6.1: Plateau's problem for minimising a surface area involves finding a surface function. A soap film can solve the problem by assuming a surface of minimal energy

6.2. Genetic Programming

Genetic programming (GP) is the automatic generation of computer programs by computer programs, and therefore requires moving to a higher level of abstraction. As such, instead of investigating the implications of altering parameters to explore the possible outcomes of a single computational process, GP investigates altering entire processes themselves *as well as* the lower level parameters. GP is very closely related to Artificial Embryogeny, because computer programs that generate computer programs can be seen as analogous to evolutionary development, that is, the evolution of the process of developing form (see Chapter 5.2). In GP, explicit embryogenies are commonly produced because the developmental process that are evolved can be understood, either in the form of computer code or as a visual graph or tree structure.

Werner (2001) draw parallels with GP and the development of the calculus of variations in mathematics. Whereas traditional differential calculus is concerned with finding the maxima and minima of functions, variational methods search for mathematical functions themselves using *functionals*. An early example in variational calculus is Plateau's problem from 1760 where one must find the function of a surface with the minimal surface area given a boundary condition (fig. 6.1). Both GP and variational calculus share the same underlying principle of broadening the scope of the solution space beyond numeric variables.

6.2.1. Tree Representation

Cramer (1985) first developed an adaptive system for generating mathematical functions with an *explicit* tree representation which were then evolved using genetic algorithms to solve specific problems. Computer programs are however more general than mathematical functions. Koza (1994) went on to develop Cramer's ideas further by utilising the tree

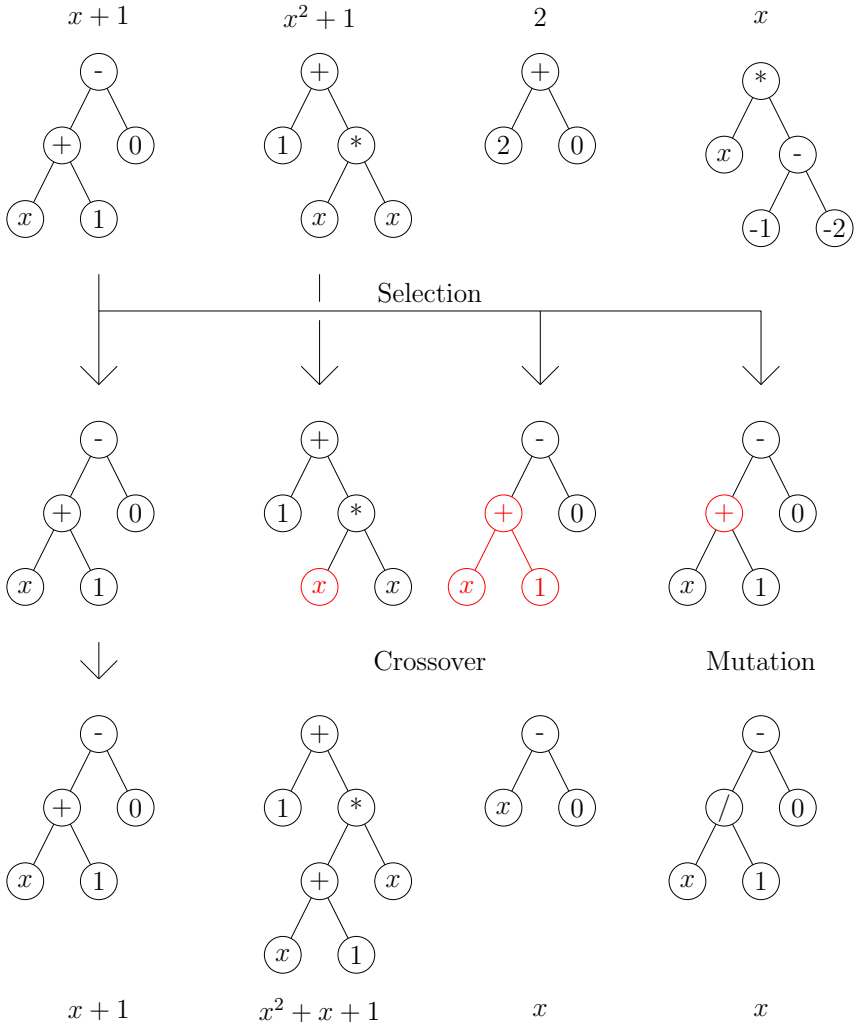


Figure 6.2: A classic GP example. Operations are organised in a tree structure, with terminal nodes formulating a mathematical equation

structure of the LISP programming language, in doing so established GP as a serious research discipline. Here the *genotype* is the tree itself consisting of nodes with mathematical variables and operators, and the *phenotype* the resulting function from this grammar. The classic operations of a genetic algorithm such as selection, crossover and mutation are applied directly on this structure (fig. 6.2). The term *genetic programming* infers that a naturally inspired metaheuristic is used to breed the programs, however (confusingly) the term GP can encompass the use of other search methods such as Simulated Annealing (Miki *et al.*, 2007).

The explicit mapping between genotype and phenotype with Koza's tree representation, means that 'bloat' is a commonly recognised problem. Tree structures will tend to increase in size and thus cause problems when computing certain programs. As Poli *et al.* (2008, p.101) state:

> "Bloat is not only surprising, it also has significant practical effects: large programs are computationally expensive to evolve and later use, can be hard to interpret, and may exhibit

poor generalisation"

The problem of bloat was acknowledged by Sam Joyce and myself when first outlining the possibility of a Meta-Parametric approach as opposed to using trees (Harding *et al.*, 2013). As we will see later on, this means incorporating an upper limit the number of components on a generated DAG structure and hence control bloat.

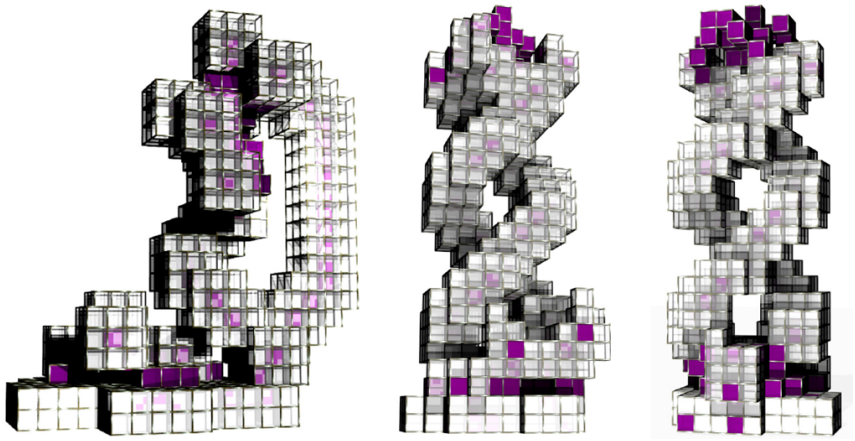
6.2.2. Generative Grammars

In linguistics, a generative grammar system is an explicit definition of how words are put together to form sentences and hence language. Noam Chomsky (1957) showed how natural languages could be decomposed into their syntactic components whose structure represented trees similar to those used in LISP (see above). Language was therefore a combination of low level rules that arranged words whose resulting structure from those rules which had a tree representation known as parse or syntactic trees. Syntactic trees include words and the structure lays out their combination *explicitly* which then forms a linear sentence (Fischer, 2001).

It is natural to extend the idea of a grammar to geometry. For example, we can think of a parametric schema constrained to a tree structure as a geometry language made up of component functions (words) with certain methods of creation (rules) and edges (location in a sentence). In computational geometry the data type of the primitives (Point, Line, Surface) have similar logical rules associated with their creation set out by their constructors (Point by XYZ, Line by Points, Surface by Loft, etc...). Although not popularised in mainstream parametric modelling software, the field of *shape grammars* has already explored architectural applications of generative grammars.

Generic Semantics

The idea of automating parametric model generation can in some ways be seen as similar to shape grammars that set to establish an architectural language which is then assembled combinatorially (Stiny *et al.*, 1978). Such systematic approaches to design attempt to emulate architectural expression and add semantic meaning to certain geometric shapes and processes in a reductionist fashion before the design process begins (Alexander *et al.*, 1978; Mitchell, 1990). However, as Derix (2008, p.23) states, this means that "shape grammars do not provide any alternative to the algorithm; the expressions are already embedded in the description."



production rule 2 | context sensitive | operand —> F | left context —> "G" | right context —> null | sequence —> F-^F+&-+-G[^F+G-[&F^|G-^FG]&FH[-^GF[FGGF^-|FFG|+&F^|^GFHF+|GGF+GG|HF|FF|H-^^|F^|HFF]&FF[F-F|^F&F+H|HHFF]FF+FF[&F]F+--|HF[&FFF^-]-G^G|H+G[HFFFG|GFFFF|-&F|FF]^+FFFF|+&FF+&FF+&FF

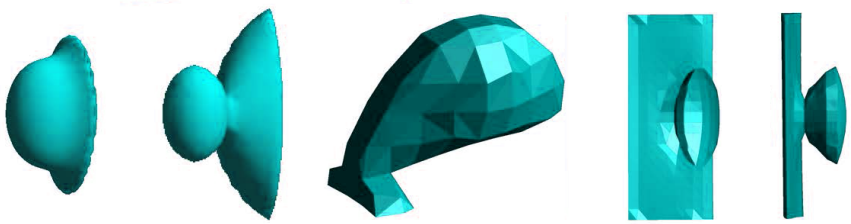


Figure 6.4: Example objects evolved using a CPPN. Source: (Clune & Lipson, 2011)

explicit embryogeny that included recursive structures similar to Lindenmayer Systems was used to generate tower forms (Senatore, 2009) (fig. 6.3). Again, Senatore found that the direct mapping of a formal grammar helped with the evolvability of the generated phenotypes, and a tree representation could also be generated to assist with human legibility. Coates (2010) has also investigated the use of simple geometric boolean operations arranged in trees as part of his GP Dom-ino project, although these take the form of tree structures used by Koza and not DAG structures as used with parametric models.

Evins (2012) has recently investigated the evolutionary development of building forms for multi-objective problems. This work uses the Compositional Pattern Producing Network (CPPN) representation proposed by Clune and Lipson (2011) (fig. 6.4). The CPPN uses combinations of simple mathematical functions similar to Koza's LISP trees in order to map point locations to a voxel boolean state. Although a wide number of forms can be developed using CPPN and high evolvability can be achieved due to its explicit formal logic, it was not clear how such a process would be integrated into existing collaborative parametric modelling systems, the motivation behind this work.

6.2.4. Cartesian Genetic Programming

Similar to Coates' ideas on the Dom-ino project (2010) that were limited to tree based structures, the question of how to replicate a similar process with a parametric model DAG is a compelling one. In computer science, as well as Koza's tree structures that represent computer programs, the automatic generation of directed acyclic graphs was first attempted shortly after due to their close relationship to computer algorithms (van Leeuwen, 1991) and historical use in project management known as the Program Evaluation and Review Technique (PERT) (Boulanger, 1961). Such automatic DAG generators tended to have complicated encodings and never (until recently) found widespread application.

Fundamentally, any DAG representation (with Grasshopper no exception) can be rewritten as a tree, however the nature of the DAG gives several advantages. Firstly, the combinatorial logic means that outputs

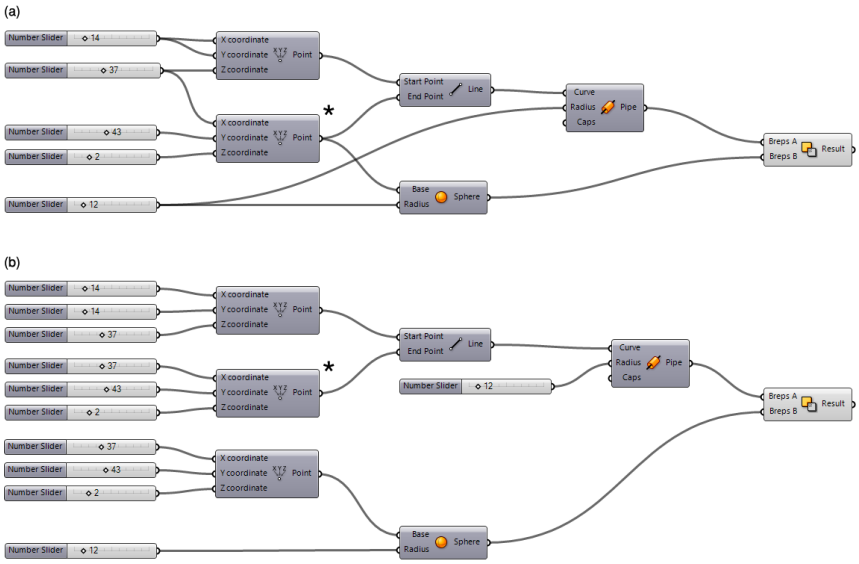


Figure 6.5: Grasshopper definitions as a DAG and as a tree that give an identical outcome

can be used by more than one input, increasing the likelihood of element reuse. Complex structures can therefore be created with less information and computed in less time Woodward (2006). This is highlighted in the example shown (fig. 6.5), whereby a simple DAG Grasshopper definition (a) gives an identical outcome to a tree with repeated elements (b). The DAG representation reduces the dimensions of the solution space, from ten to five, and crucially builds shared dependencies in the model altering its variability. In the example, both the sphere and the pipe share a dependency with the point (highlighted *).

A DAG interpretation of L-Systems is described by Boers and Sprinkhuizen-Kuyper (2001) similar to a another method known as Cartesian Genetic Programming (Miller & Thomson, 2000) (CGP). The latter has recently gained popularity due to its apparent ease of use and integer based encoding. This process generates explicit structures very similar to those used in parametric modelling systems such as Grasshopper. Primitive functions are housed in component boxes which are then associated to each other in a dependency hierarchy. The *Cartesian* term simply means that each component box has a presence on a two-dimensional canvas as a form of visual representation.

A typical CGP set up is shown in fig. 6.6. A single integer string genotype is capable to storing metric, topological and functional information that make up each unique DAG that is computed to formulate a set of equations. Modifying the genotype means the resulting phenotype is not just changed in terms of the parameters, but also the structure of the graph itself.

Crucially, because the mapping between genotype and DAG phenotype

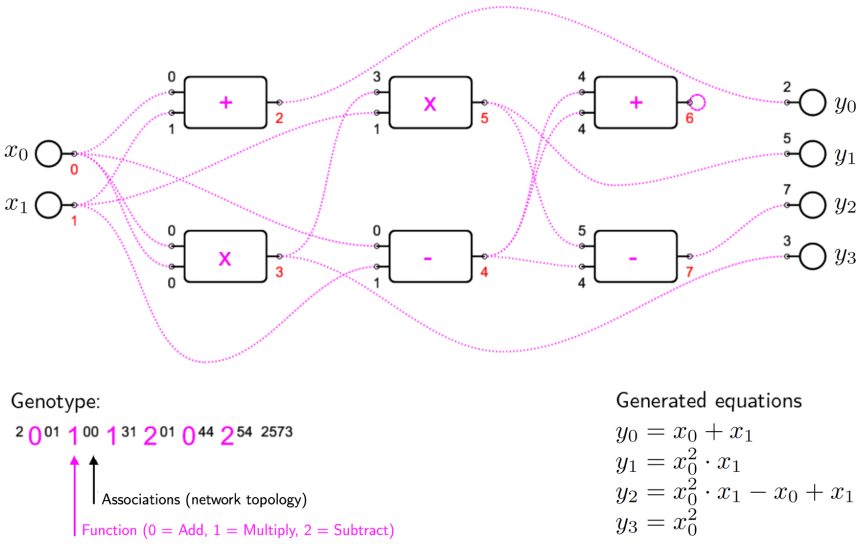


Figure 6.6: A Cartesian Genetic Programming example

is direct and explicit, better evolvability can be achieved (see Section 5.5.2) The closer relationship between the genotype and phenotype (low epistasis) leads to faster adaptation to the current problem. CGP has been successfully implemented for the evolution of electrical circuits in combination with genetic algorithms (Miller & Thomson, 2003; Vassilev & Miller, 2000). In addition, CGP has been shown to be easily implementable in a wide range of fields. These include machine learning, neural networks, data mining, financial prediction, function optimization, classification, medical diagnostics and evolutionary music. It is not a big leap to see how Cartesian Genetic Programming could be applied to parametric models, increasing their topological flexibility and therefore making them more suitable to the conceptual design stage.

6.3. Embryo

6.3.1. Introduction

Embryo is the name given to a plug-in to Grasshopper written by the author in C# that combines genetic programming with parametric models. The approach involves using a similar mapping method to CGP for encoding a parametric model schema, thus enabling its structure to be agile as part of a wide design exploration. To the author's knowledge, no attempt to generate parametric model definitions automatically has been previously attempted.

Instead of developing an autonomous stand-alone application, it was purposefully decided to develop software that would work within an existing environment as a plug-in. Rhino Grasshopper was chosen first

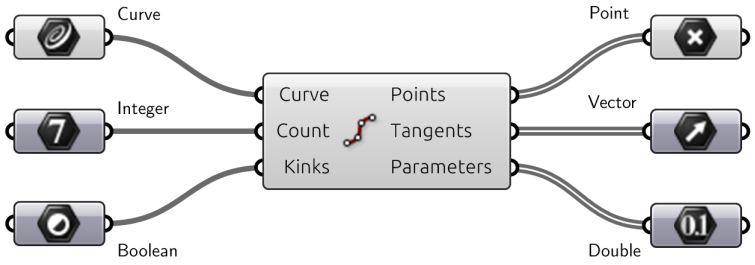


Figure 6.7: A typical grasshopper component defines its input and output parameters and their preferred data type

because of its current popularity as a visual programming environment, and hence a meta-parametric approach would be akin to a trojan horse, attempting to modify the process of model generation from the inside. Secondly, due to its geometric primitive components having an explicit single constructor, for example 'line by points' is a separate component to 'line by origin, direction and length' (fig. 6.7).

This meant each component encapsulates the *grammar rule* behind how a geometric primitive is created, and not just its final presence in the model. Many additional third-party components are also available for Grasshopper and as of writing, these are increasing in number. In reality, any parametric modelling software could be used that utilises a graph based approach and many of the principles developed here would equally apply to other parametric software (such as Bentley Generative Components or Autodesk's Revit with Dynamo).

6.3.2. Generating DAGs

Embryo generates DAGs automatically by allowing the user to work at a higher level of abstraction. Instead of defining the graph top-down, the user can define the conditions that limit the graph definitions generated. These include what components are to be included, the number of components, the number of numeric parameters, the parameter domains, etc... As with CGP, a numeric genome encodes the generation of a graph structure from these conditions. Once the system is prepared, the generation of different DAGs from base level components is simple, and when calculated, different forms are easy to explore quickly.

A typical Embryo set up in Grasshopper is shown in fig.6.8. The grasshopper canvas is now divided up into separate areas. The main Embryo control component sits on the usual place in grasshopper, able to be manipulated top-down by the user. The input components (or ingredients) that will make up the generated graph are located off the main canvas to the left. The generated parametric model is above the main canvas and includes metric parameters, components and the topological structure of the graph. Embryo can include existing components (as well

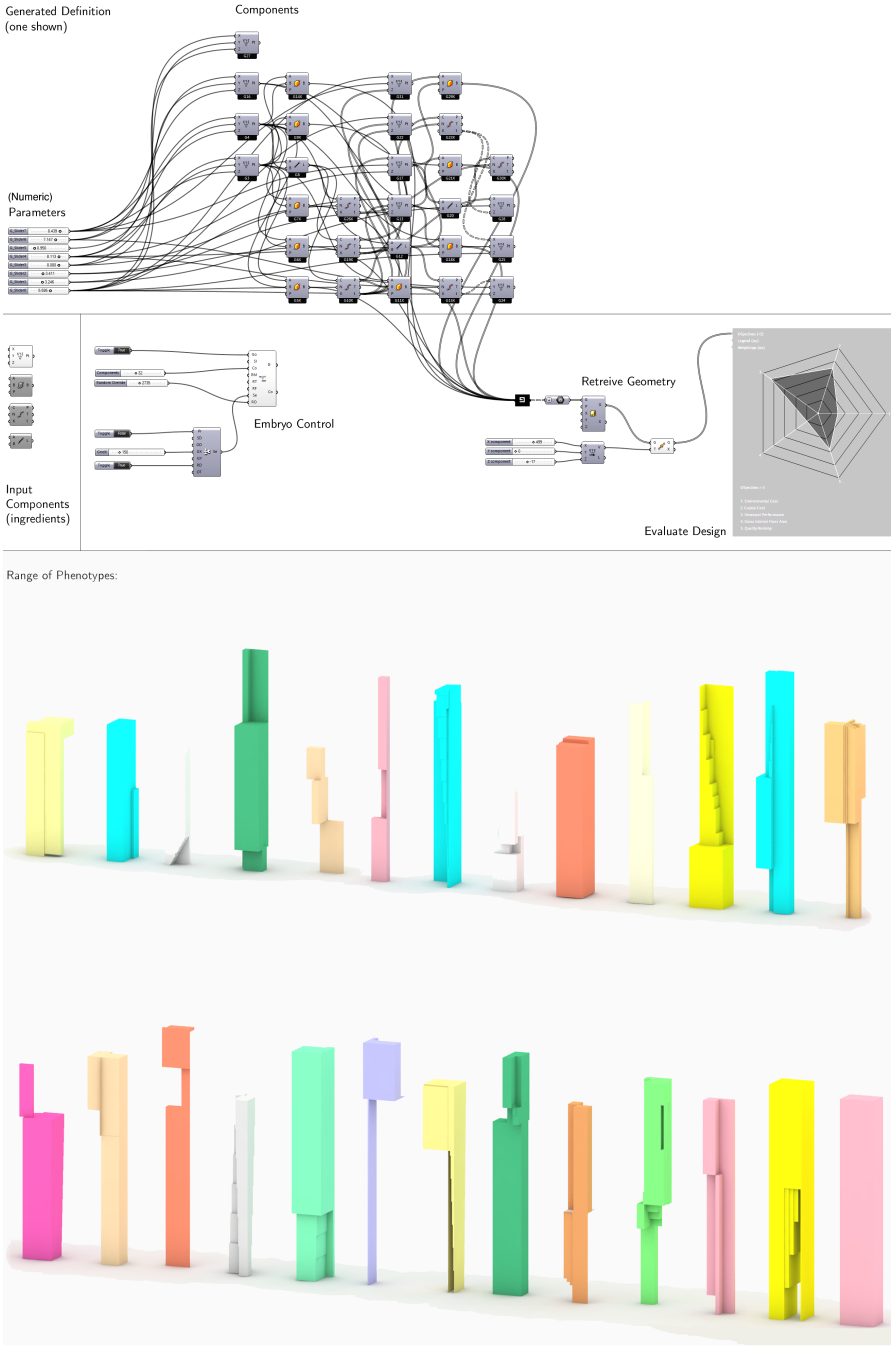


Figure 6.8: Typical Embryo set up in Grasshopper generating simple tower forms

as generating new ones) by simply dragging them above the main canvas before the process is run.

The outputs from generated components can be referenced back to the main canvas shown on the right. This geometry can be used in any usual Grasshopper definition, for example in visualising performance as shown in the example. So long as the geometric expressions housed in components are sufficiently primitive, the design domain can now be explored in a more healthy manner as topological inflexibility can be released during a search process.

6.3.3. Embryo Components

As a plug-in to Grasshopper, Embryo exists a series of components that can be placed on the main canvas and control the nature of the generated graph definition (fig. 6.9). These are divided into the following five categories:

1. Parent components that are used on the usual Grasshopper canvas. These include the main Embryo controls, settings component and three components used to tag outputs and inputs to be included in the generated definition.

2. Child components that can be added to the generated graph. Currently this only includes a single component to block outputs of the generated definition.

3. Cognition components that assist with human understanding of the generated graph. The use of these components are discussed further in Section 6.3.8.

4. Utility components - non specific components including one to reveal the data type of a component output and another to unfold a design to a net for physical model making.

5. Visualisation components - a series of components that display charts for viewing performance data of the generated models in real-time.

6.3.4. Constraints and Parameters

As well as creating their own parametric models top-down, the design team can now work at a higher level of abstraction by manipulating the Embryo component. The main limitation that Embryo currently imposes is that each input has only one incoming edge. It is hoped that future versions will address this issue as it has implications such as the

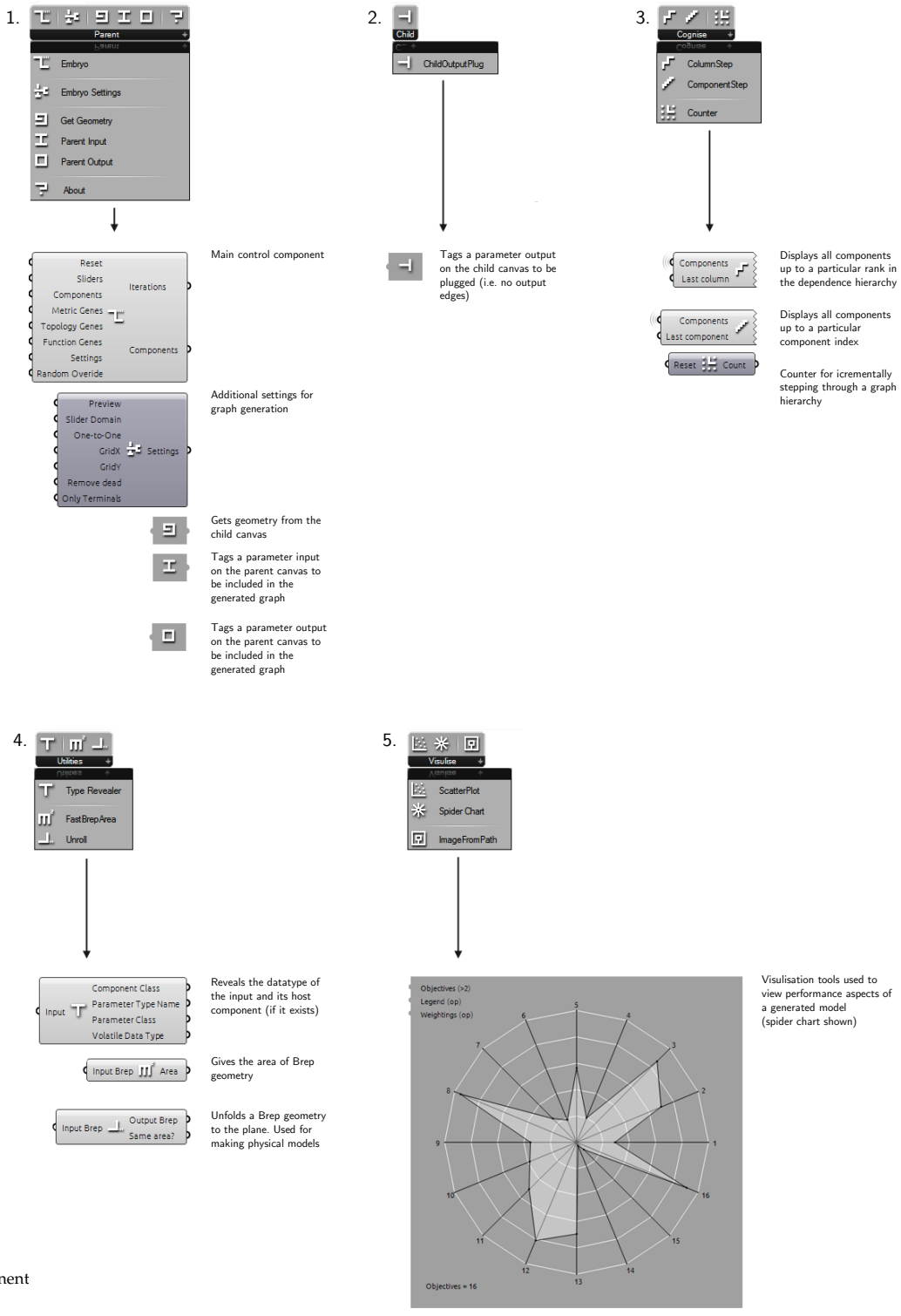


Figure 6.9: Embryo component list

impossibility of list structures being formed as inputs to a parameter. One example in Grasshopper is with the Bezier Curve component that needs a list of different input points instead of just one.

An additional built in constraint avoids the creation of null data, for example zero-length lines. This helps to cut the amount of error strewn parametric models and is specific to Grasshopper. Outside of these constraints, the design team can control several settings options to manipulate graph generation including the following:

- Number of Slider Parameters (integer)
- Number of Components (integer)
- One-to-one: an output parameter can/cannot be included more than once (boolean)
- Grid size of displayed components (integer)
- View failed components (boolean)
- Preview components (boolean)
- Display only terminal components (boolean)

6.3.5. Encoding and Generation

The generation of the graph is a direct mapping from the genotype to the graph, and similar to CGP the explicit grammar includes numeric, functional and topological information. This is reflected in three inputs for the main Embryo component as shown in fig. 6.10, Metric, Function and Topology gene inputs for which Grasshopper 'gene pool' components (shown pink) can be used as list inputs. In addition, the 'Random Override' input can be used to override these three genes and create a random graph from an integer seed - this is useful for initially exploring a solution space as we will see later.

Assuming the random override is not used, the three inputs that control how the graph is generated as follows:

1. Metric genes control the parameter values for the sliders and have a direct numerical mapping. These can be integer or floating point values and the number of numeric parameters are generated are in accordance with the settings specified by the user. These metric parameters are the first things generated by Embryo.

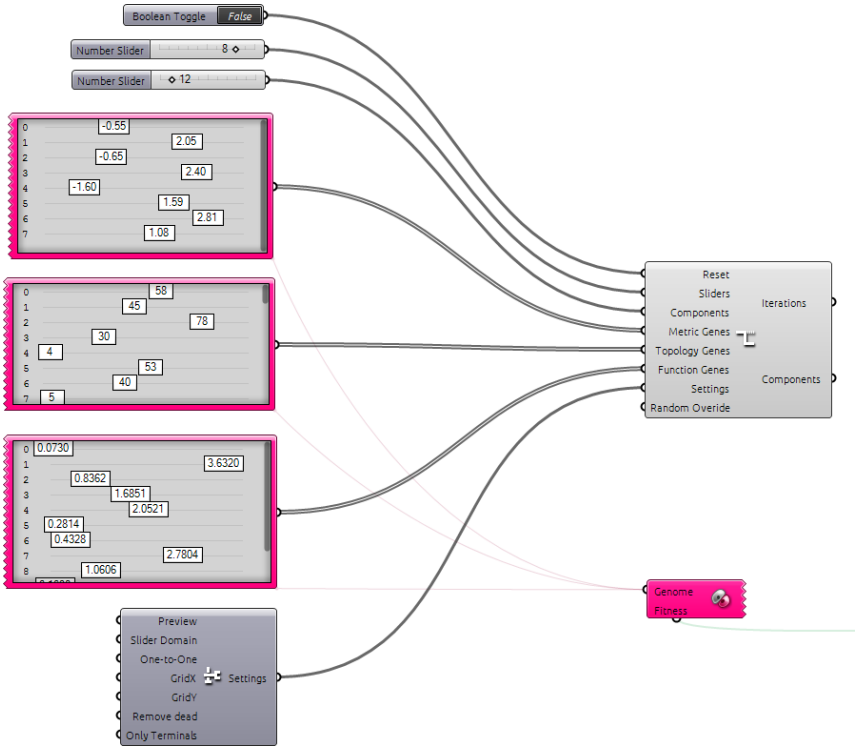


Figure 6.10: The main Embryo component with metric, topological and functional genes coming from separate gene pools.

2. When a component is added to the graph, the functional genes control which type of component is selected from the ingredients pool.

3. The topological genes are integer based and map the output location for each input when forming the graph. Altering these genes therefore changes the topology of the graph structure generated.

Figure 6.11 shows how a simple graph is generated step by step using the three sets of genes. The numeric parameters are generated first and their outputs labelled. Components are then added by using a functional gene to select its type (shown in blue), and the topological genes (shown in red) to connected each new component's input to an existing output. In the example shown, box component that requires two points as inputs is selected, however as there are not enough outputs at this stage of a suitable data type, the component is removed from the canvas. Similar to CGP, this generation method ensures that the graph is always a DAG and therefore non-cyclic, able to be computed as a new solution by Grasshopper as with a standard parametric definition.

Algorithm 6.1 shows in more detail how the graph definition is achieved by Embryo. The *outputList* stores possible outputs that can be connected to by any new component. When a new component is created, its outputs are added to this list so that they may be used by any future components. The dependency rank is stored to sort the graph into dependencies to attempt to make it legible to the human (see Section 6.3.8).

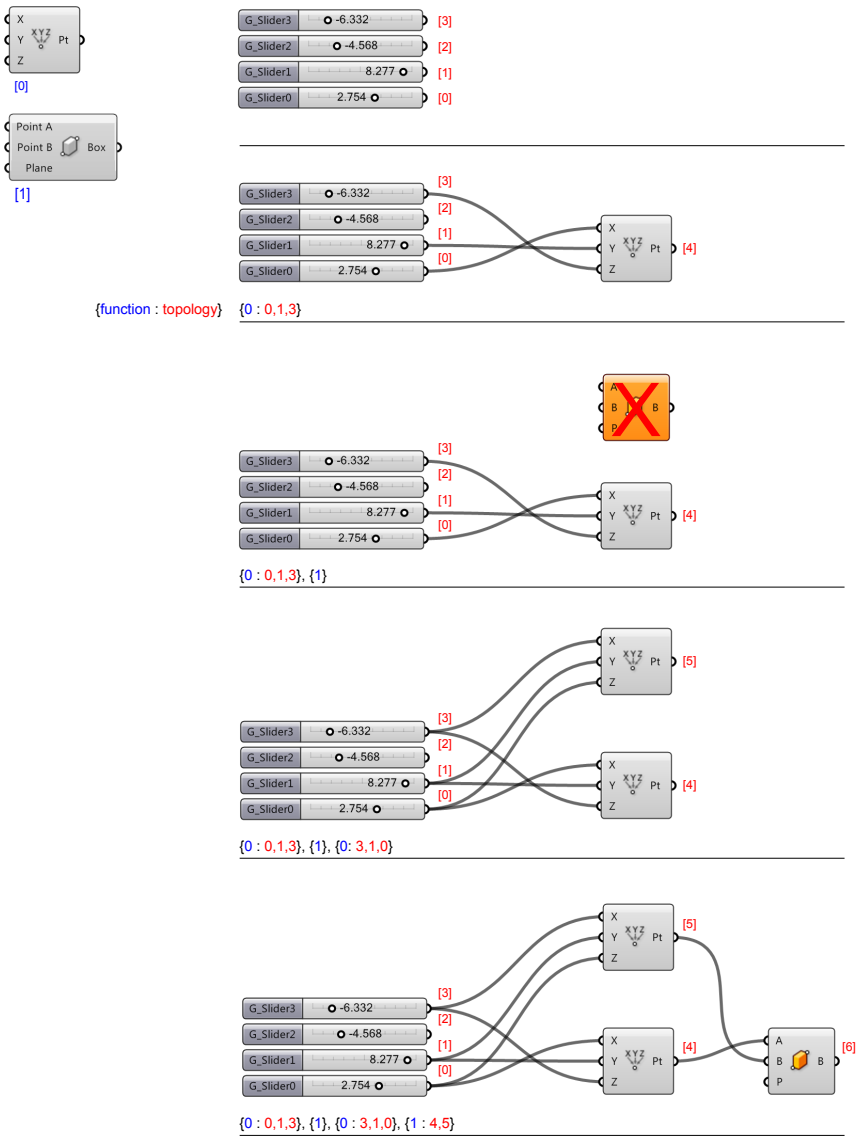


Figure 6.11: Graph generation process. Numeric parameters are generated first, then the components using functional and topological genes

Algorithm 6.1 The DAG generating process used by Embryo

```
object[] outputList
for all newComponents c do
  c.selectIngredient(functionGenes_c)
  for all c.Inputs i do
    bool typeCheck = false
    while !typeCheck do
      Inputs_i.selectOutput(outputList, topologyGenes_c*i,int k)
      typeCheck = Inputs_i.datatypecheck(outputList_k)
    end while
    int Rank = Inputs_i.DependencyRank(outputList_k)
    if Rank > c.DependencyRank then
      c.DependencyRank = Rank
    end for
  if c.warning = null then
    outputList.add(c.outputs)
  else
    canvas.remove(c) //remove the component if it failed
  end if
end for
topologySort(newComponents) //arrange the final graph
```

When a new component input selects an existing component output, the data type check makes sure that the connection is not meaningless. The syntactic rules of the parametric grammar are specific to the data type of the parameters and have a direct correlation to the constructor. As previously mentioned, Grasshopper is useful for this task because the rule system is explicit in the component itself; for example, a 'line by points' component can use a Grasshopper Point data type input, but not a Number or a Surface one.

The data type check is done with component output type and not volatile data types at present due to speed. In Grasshopper, volatile data is that which flows through the model when the DAG is computed, but to compute the graph after every additional component is added or edge is connected is comes at high computational cost. This means that the Grasshopper in-built casting procedure is ignored and instead a look-up table was created by the author that compares output parameter types. Whilst this can sometimes result in failed components, it does means the DAG does not have to be computed until the end of the graph formation process and hence is much faster.

6.3.6. Some Simple Examples

Even for just a small selection of components, the combinatorial possibilities are huge (Harding *et al.*, 2013), however the solution space of possible definitions with given conditions can be explored. In its

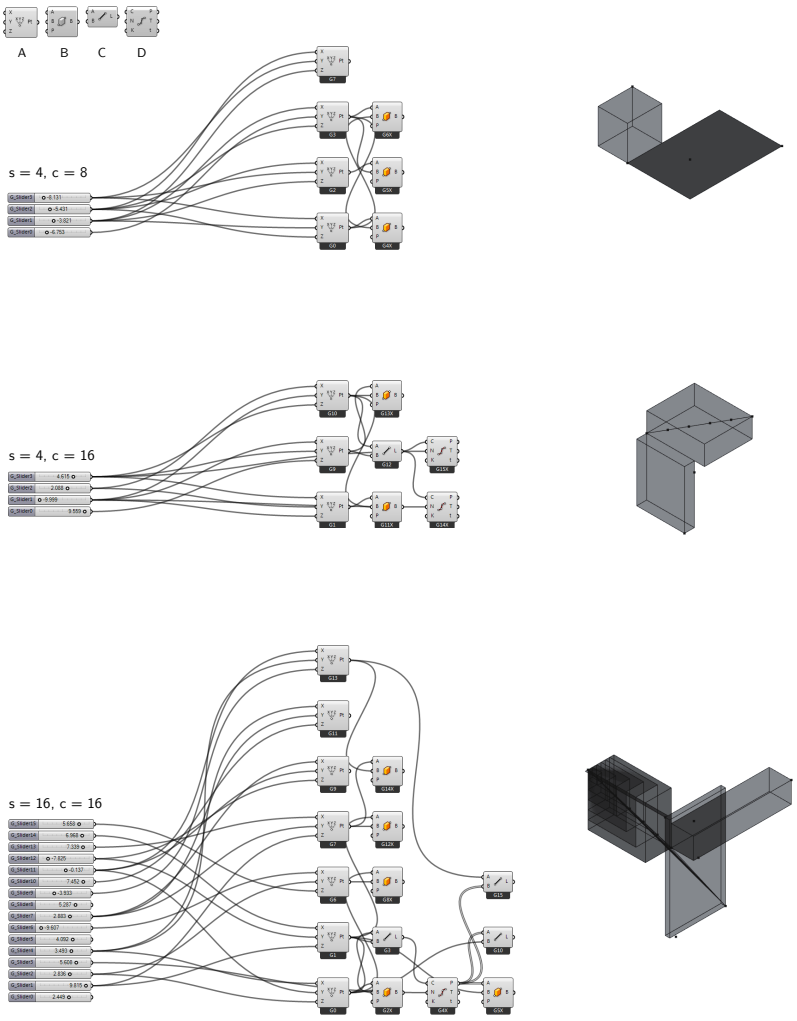


Figure 6.12: Three generated graph structures. 's' represents the slider number and 'c' component number (some have been removed upon failure)

simplest form of user interaction, the random override mentioned in the previous section can explore the solution space quickly by randomising the gene pool. Figure 6.12 shows some examples of randomly generated parametric definitions made from four components: A) Point by xyz, B) Box by two points, C) Line by two points, D) Divide curve. Further examples of graphs generated with these 4 primitive components can be found in Appendix G. We can see by visual inspection how the complexity of the model increases with the complexity of the graph, both in terms of the number of numeric parameters and components. The interaction of the user now takes place at the level of selecting a suitable set of components, adjusting the size of the generated graph and finally choosing the random seed that generates the parametric model.

The phenotypes generated by the parametric model are limited by the modelling environment itself, i.e. Grasshopper. Figure 6.13 shows resulting phenotypes from such relatively small sized graph structures generated from just a few different component sets. These geometries are all generated almost instantly by Embryo, with control shifting to the

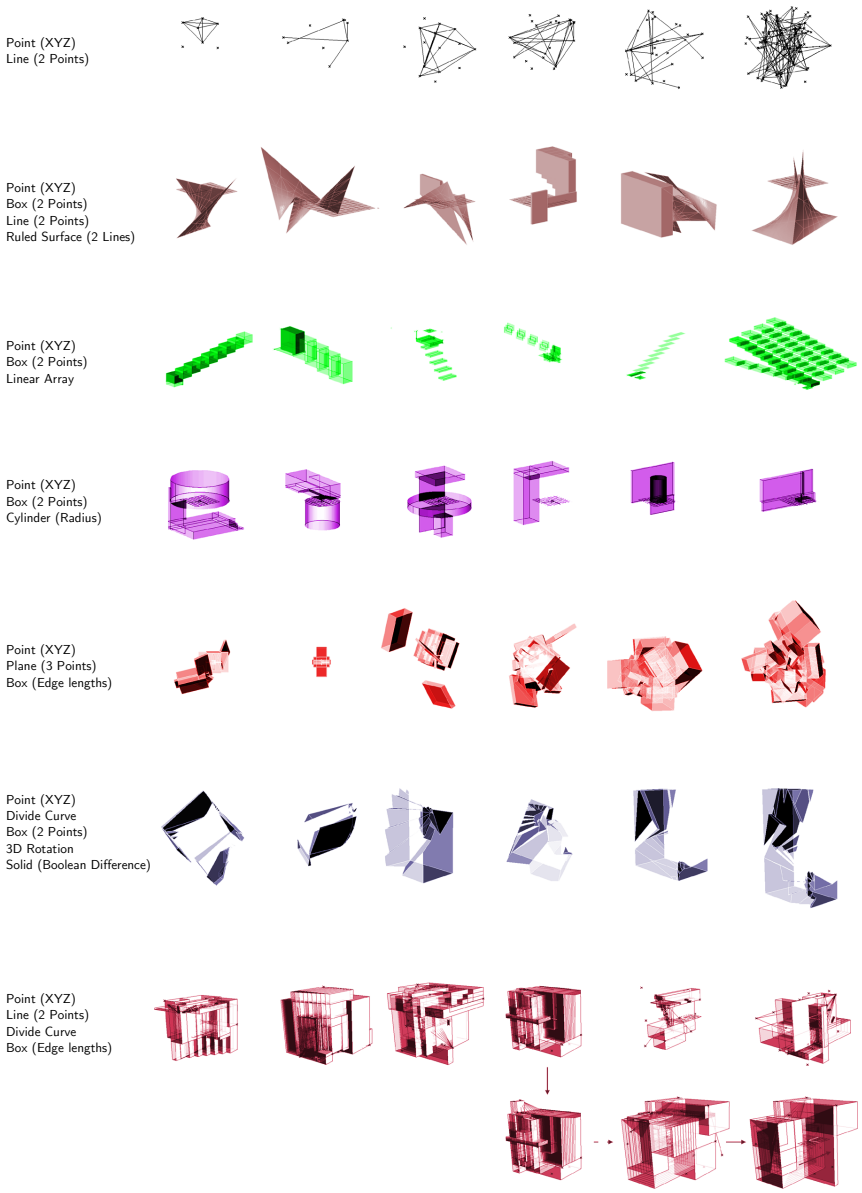


Figure 6.13: Phenotype examples generated from different component functions (ingredients). The bottom row shows manual parameter adjustments to a chosen form following initial generation.

selection of appropriate components and tinkering with the Embryo set up parameters, and away from direct top-down parametric modelling.

Although the resulting forms are made only from simple boxes, we can see from visual inspection how the variety goes beyond the adjustment of parameters alone, with various different topological structures (or body-plans) being generated both in the genotype and the phenotypes. The speed of generation is also rapid for this example, with designs available within seconds of opening a blank Grasshopper canvas. As we have seen, this is important at the conceptual design stage where many possible design directions need to be investigated in exploring a wicked problem.

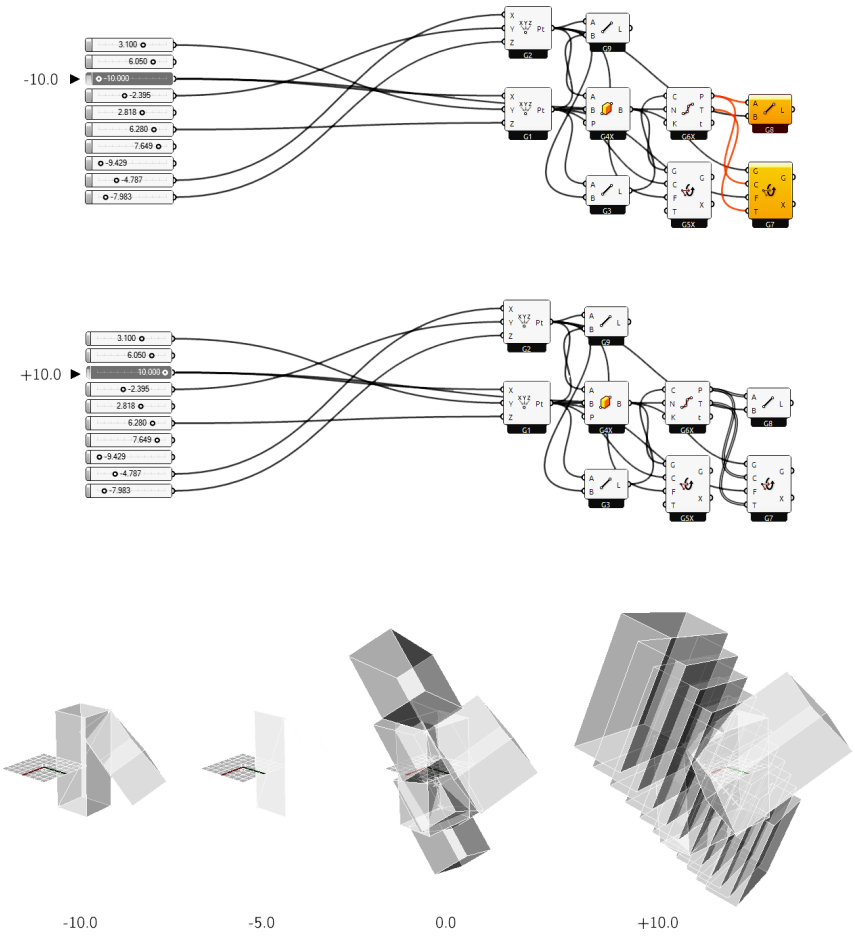


Figure 6.14: Junk DNA being used in the next generation: Two failed components come alive for the same graph when single parameter becomes positive

6.3.7. Combining with Metaheuristics

Figure 6.10 showed how the main Embryo component takes three separate gene pools for metric, topological and functional aspects of the graph. The evolutionary solver Galapagos can be connected to the genes; indeed, metaheuristic solvers can be incorporated with Embryo. The mapping process is direct, similar to Koza LISP trees that have high evolvability success and a wide variability (Wagner & Altenberg, 1996). A random override input is however provided in case the user wishes to simply randomly generate graphs and get a feel for the solution space.

If a metaheuristic solver is incorporated, then this will govern development. For example, single point crossover of topological genes using a genetic algorithm will result in completely different graph structures.

In Section 5.5.2 the importance of an explicit representation was discussed for evolvability if used as part of a feedback process such as natural or artificial selection. By eliminating bad solutions and promoting good (as opposed to finding the best), parametric definitions that generate meaningless outcomes are less likely to remain, though not eliminated completely. There is a parallel with natural systems that carry

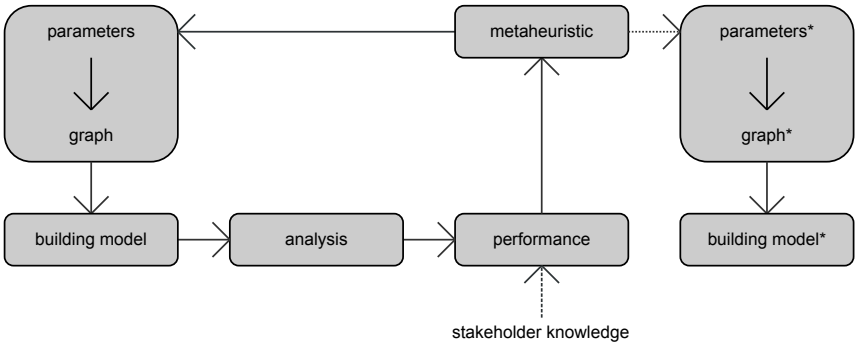


Figure 6.15: Metaheuristic algorithm operating on parametric model including its graph structure

junk DNA that maybe utilised at a later generation to maintain diversity, and in CGP is also known as the propagation of schema or *neutrality* (Miller, 2001). The influence of neutrality in CGP has been investigated in detail and has been shown to be extremely beneficial to the efficiency of the evolutionary process on a range of test problems (Miller & Harding, 2008). As an example, fig. 6.14 shows an example of a parametric model that contains *junk* components that at the next iteration come alive. Here the change in the graph is minimal but the parameter values have a large impact on the resulting phenotype when one becomes positive.

In combination with existing performance analysis tools, computational analysis can now provide decision support for various building types and not just those set by an initial parametric concept. As well as generating parametric models one-by-one, due to the encoding of the graph structure into a string, the generation of the parametric models can be combined with a metaheuristic algorithm such as genetic algorithm or simulated annealing. fig. 6.15. An example of combining Embryo with a metaheuristic is given in Section 7.3, where parametric model is found that defines a target geometry.

6.3.8. Cognition

As we have seen, although the generation of forms is automated with such a system, crucially each design also has an explicit parametric definition which can be visually inspected and integrating into an existing model. Embryo generates a DAG that is already topologically sorted. This means the dependency hierarchy and hence how the final geometry is constructed through a series of algorithmic steps can be better understood by the user. Interestingly, attempting to understand how the machine has constructed the final model is actually the same as understanding how other humans and hence other consultants have gone about their parametric models. As we saw in Section 4.2.2, the *legibility* of the graph is an important aspect to a parametric model for good collaboration. As with coding standards for clean collaborative

code in software development, so parametric modelling standards can help in communication of not just final forms, but *how* they were arrived at and at what scale each geometric operation occurs.

Derix and Izaki (2013, p.44) state that "by visualising the processing of an algorithm step-by-step, its components and data in a diagrammatic near-notational form, the logic and behaviour of the system become transparent". Although Grasshopper restricts interaction only at the level of the graph, the model can still be observed simultaneously when attempting to understand a particular model. As such, cognition of the computer's geometric process doesn't just mean having to untangle the spaghetti, but also just observing the model construction in the model view as it is built up step by step. This encompasses two ideas behind Generative Components (GC) (Aish & Woodbury, 2005), in that there should be more than one mode of representation for the process of form development and that the linear build of the model itself can be recreated and thus explained, a process in GC known as 'transactions'. As opposed to Grasshopper when one is presented with the entire graph 'as is' upon opening, GC features the undervalued transactions concept whereby discrete pieces of logic are recorded at key stages of graph development. Here, a stakeholder unfamiliar with the model can step by step go through its construction in a sequential manner, thus replicating its construction and previous modifications.

Embryo employs a similar concept to GC's transactions by making transparent the code that has evolved (i.e. the code that generates the building), but in a structured hierarchical order by conducting a topological sort of the parametric schema. The cognition set of tools then allow the user to run through a model step by step and therefore attempt to understand how it is been constructed (fig. 6.16). Although the topological sort and step by step previewing used by Embryo can help matters, there is however no escaping the fact that spaghetti is still created and will require effort on the part of the user to understand. In response, one could imagine some method of *untangling the spaghetti* according to a standardised procedure similar to the graph planarity problem. Indeed, manipulating a parametric model sometimes feels like trying to solve John Tantalo's online planarity game (Verbitsky, 2008) whereby increasingly complex graphs have to be untangled. Although not covered further here, this would be an interesting unexplored area to address using Embryo's cognition tools.

In addition to the topological layout, Embryo also has user options to help with graph cognition, such as removing failed components (junk DNA) if required just for visual purposes, changing the spacing of the component layout and the option to only display terminal components.

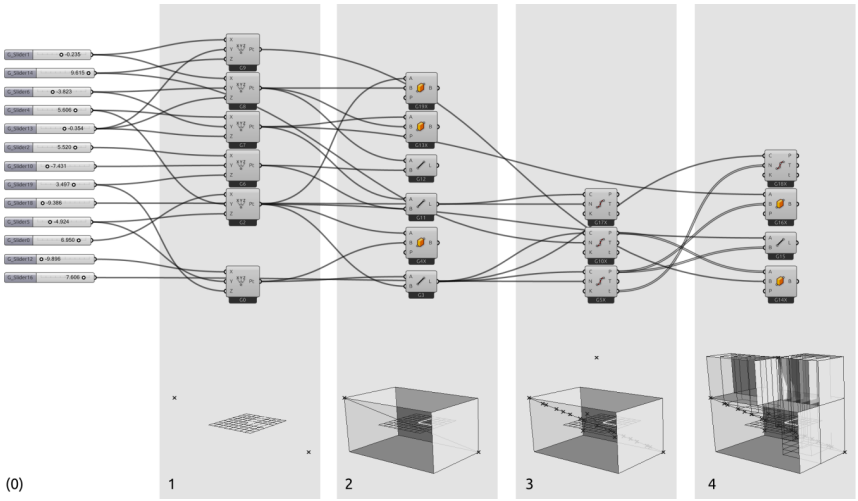


Figure 6.16: Embryo creates a topologically sorted parametric model which can be previewed step by step. This reveals hierarchical dependencies and explains visually how the final form is constructed to the user.

Finally, the choice of component primitives will influence the cognisability of the visual program once generated. For example, it is true that a whole geometric process can be compiled into a single component, and therefore be somewhat inaccessible for understanding. It is therefore the responsibility of the user to deploy which set of components he/she thinks will generate a sufficient number of designs to suit the specificity of the design problem. The important thing is that the design team is able to decide this for themselves and is not locked into a set number of functions created by others.

6.3.9. Working In-Between

The computational design community can arguably be divided into two camps. The first, those concerned with top-down parametric modelling or computer programming. Second, those concerned with procedural modelling methods based on bottom-up rule systems. There is no right or wrong approach, and design often requires both of these aspects under one system. Human designers form their individual design experiences into generalised concepts or groups of concepts at many different levels of abstraction (Gero, 1990). By deploying Embryo on an existing parametric software package, working between both top-down and bottom-up generative processes becomes possible for the reasons given at the end of Chapter 5.

We will see in Section 7.1 an example whereby Embryo was able to incorporate an existing parametric process and therefore mix top-down and bottom-up methods of visual program development. I believe that this inter-changeability is crucial in allowing such an approach to be used at various stages of design development. Design works at multiple scales and sub-problems tend to be resolved not at once but piece by piece,

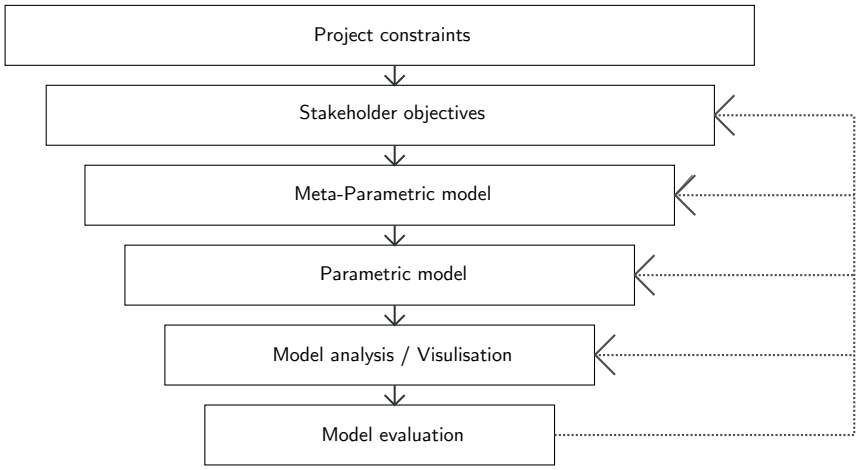


Figure 6.17: Layered model showing visualisation and subsequent interaction possible at various levels of abstraction

hence one must consider the effect of scale and dealing with complexity at different levels of hierarchy (Simon, 1962). Through a Meta-Parametric approach, the design team can work at different levels of abstraction within the building model appropriate to the current design task (fig. 6.17).

6.4. Discussion

This chapter has introduced a plug-in for Grasshopper (itself a plug-in for Rhino) based on meta-parametric thinking. Inspired by genetic programming, Embryo automatically generates parametric design models enabling exploration of different combinatorial options (and not just by adjusting numeric parameters).

By remaining within an explicit DAG representation, models automatically generated can *potentially* be understood by design teams and therefore understandable as collaborative artifacts. This claim is a hypothesis that requires testing however, and the effectiveness of a meta-parametric approach is therefore explored in the next chapter on a series of real projects. Can models be generated by Embryo actually be understood by design teams and used in an effective way? If so, it offers an example of machines working with human designers in order to explore design problems and not simply as problem-solving slaves.

In addition, can Embryo be used for target based evolution, evolving parametric models for a given geometry, perhaps even a geometry made by a different parametric structure and thus open up new avenues for exploration?

7. Embryo Experiments

> "One really interesting possibility that has been under-explored is using the exact same tools not to generate patterns, not to generate cold-blooded efficiency, but to discover things that you want that wouldn't have occurred to you otherwise. It's basically a way to enhance your creativity and to show you something new."

David Benjamin (2014, p.36)

Embryo is a plug-in for Grasshopper that aims to automatically generate parametric models with particular application at the concept design stage. In this chapter, Embryo is tested in three scenarios. The first two concern real-life projects with two different architectural practices. In both of these examples, Embryo is used as part of an early-stage design process with a mixture of known and unknown objectives. The third explores Embryo as a pure problem solver, attempting to match a target geometry with a parametric model definition.

7.1. Experiment 1: AG5 Architects

The first experiment was a collaboration between Ramboll and AG5 Architects. Based in Copenhagen, AG5 were a practice that Ramboll had already worked with previously. The Gran Rubina Tower 3 case study project [29Grt] was for the design of a new office tower in Jakarta, similar in scale and scope to the Escher Tower Project [16Esc] discussed in Chapter 4. In that example, wide design exploration was difficult due to the limitations of parametric modelling. The similarity of the two projects is useful, as it provided a good benchmark against which to test an alternative approach using Embryo in a collaborative environment.

7.1.1. Set up

We agreed to work together with the architect in exploring a wide range of tower typologies, evaluating each design against quantitative

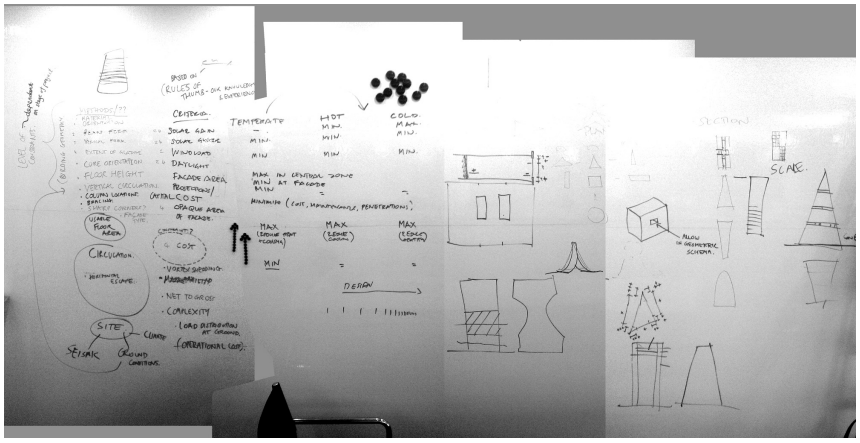


Figure 7.1: Formulating constraints and objectives on Gran Rubina Tower 3 in collaboration with AG5 architects

performance objectives alongside the architect's own evaluation methods and design processes. AG5 were slightly different to Bjarke Ingels Group in terms of their design methods, working less with physical foam models and more with CAD in combination with hand sketching. Initial meetings with AG5 at their Copenhagen office helped to establish an initial set of constraints and objectives as we knew them at the time (fig. 7.1) whilst acknowledging that these would no doubt change during the design process.

The initially identified constraints for the tower were the following:

- Site dimensions, tower location, local infrastructure, local planning authority.
- Gross external floor area (36,000m^2).
- Floor to floor height set at 4m.
- Concrete frame and core due to local labour experience and the high cost of importing steel to the region.

The initially identified objectives, six quantitative and two qualitative were the following:

- Maximise net to gross floor area (a method of estimating concrete core size was developed by Mark Pniewski at Ramboll).
- Minimise structural concrete volume (a method of estimating concrete volume was developed by Mark Pniewski at Ramboll).
- Minimise heat loss (surface area to volume ratio).
- Maximise similarity between floor plans.
- Minimise façade complexity (estimate amount of doubly curved surfaces).

- Minimise Solar gain (should be minimised in warm climates such as Jakarta, either through building form or façade shading elements).

- Create something we all feel is right (this objective evades a better description).

- Create an icon - the client required a tower that should not be the tallest in the region, but still be distinct in the region.

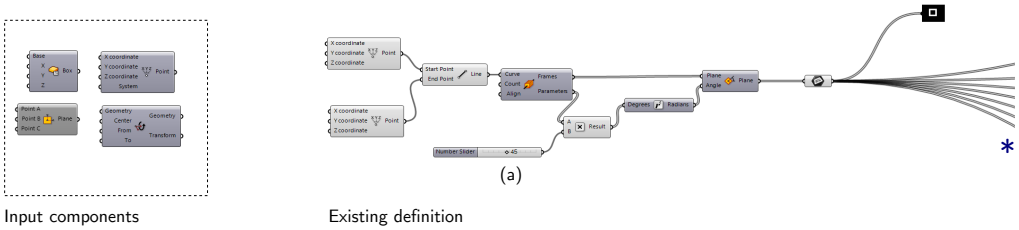
7.1.2. Modelling

At the start of the collaboration, AG5 already had some ideas that had been translated into CAD geometry, and these were to be included in any comparisons with towers generated by Embryo.

At a series of design team meetings at AG5's Copenhagen office, Embryo was used to quickly generate additional tower forms by using the random override input (see Section 6.3.5). This option negates the use of genes for each aspect of the DAG (i.e. metric parameters, functions and topology), instead generating random graphs to explore the design space. To encourage tower-like geometries, we decided to restrict some aspects of the design exploration by creating a line subdivided by planes with their outputs tagged (i.e. to be included in the generated definition). This gave geometric repetition going up the tower that could be controlled by a standard numeric parameter on the parent canvas (fig. 7.2),

A series of primitive Grasshopper input components were used interchangeably during design exploration. In the example shown, the 'Point by xyz', 'Plane by 3 points', 'Box by centre plane' and the operator 'Rotate geometry' are shown. By adjusting the existing numeric parameter (a) for this graph structure, variations for a particular tower typology could be explored *alongside* parameters generated by Embryo.

Embryo's 'GetGeometry' component (see fig. 6.9) was used to automatically pull any geometry from the generated model back to the parent canvas for further manipulation. This post-production work included scaling the form to fit the site, orientating the tower and scaling the generated model in the z-axis to meet the gross external floor area constraint. This process meant that all the tower designs that were generated met constraint requirements and could now be compared according to the objectives. Similar to the Escher Tower Project [16Esc], real time analysis could be performed but this time on a wide range of designs against the objectives indicated in the previous section. This enabled stakeholders in the design team to get a better grasp of the possibilities within the design constraints.

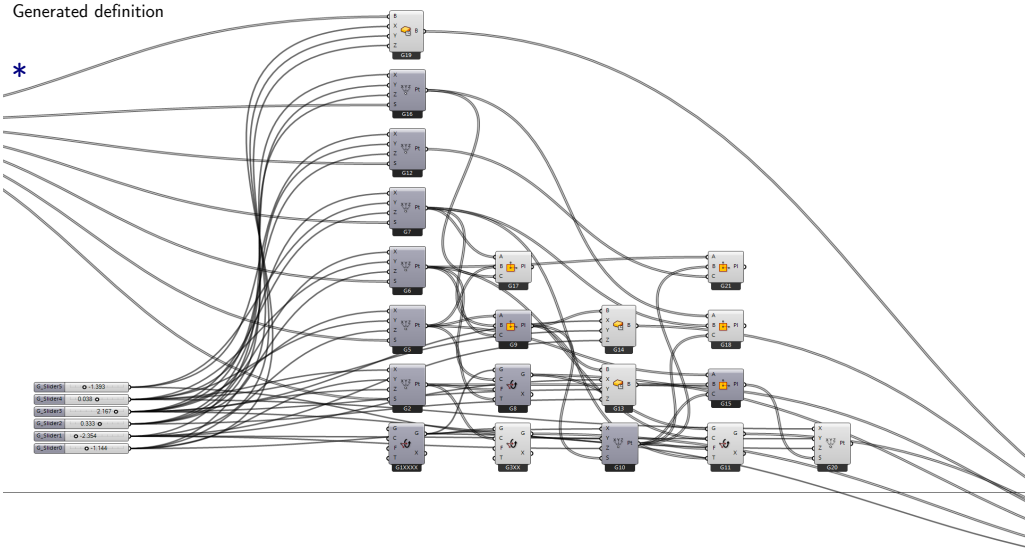


Input components

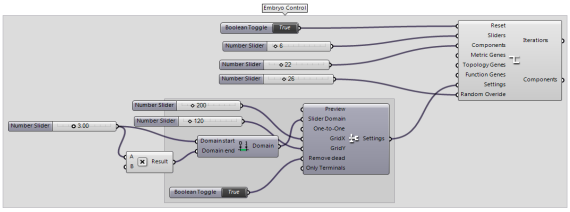
Existing definition

(a)

Generated definition



Embryo control



Post-production work

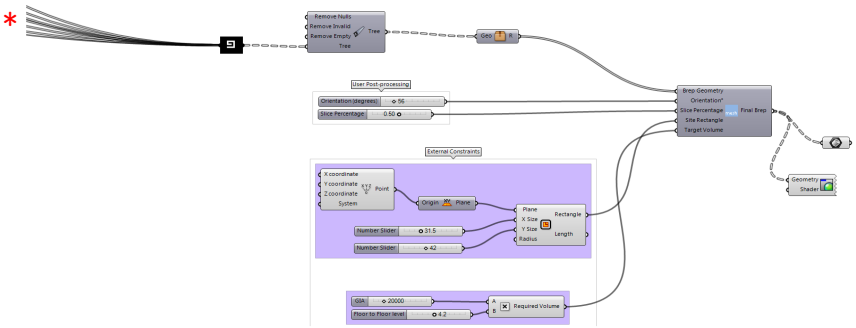


Figure 7.2: Embryo set up on Gran Rubina Tower. A particular phenotype with metric variations is shown.

7.1.3. Model Examples

Four example towers generated by Embryo are shown in fig. 7.3, referenced A to D. For each parametric model, five variations are shown made from manually adjusting the generated numeric parameters. Each of these models met the required floor area exactly and were within the site boundary constraint. Figure 7.4 shows the associated graph definitions that use four identical input components (Towers A-C) and fig. 7.8 shows an alternative set of component inputs. Each model generated is of increasing complexity, also reflected in the phenotype itself - indeed, there appears to be a general correlation between graph size/complexity and that of the generated phenotype - perhaps an expected result due to the explicit grammar of Grasshopper definitions.

7.1.4. Performance Evaluation

Following generation of a large range of different tower forms, these were stored as meshes could be assessed qualitatively and quantitatively by the design team. The quantitative performance objectives listed in Section 7.1.1 were measured using additional processes by different stakeholders. For example, façade complexity was measured by Ramboll façades team by finding the total area of Gaussian curvature above a given threshold (thus giving an estimate of required hot-bent glass). Comparison of the designs was then achieved by collating data and ranking them for each objective. Radar plots showing the relative trade of of multiple objectives was developed as part of Embryo to enable the design team to better understand how different designs compared *relative* to one another (fig. 7.5).

As well visualising multiple objectives, single objectives could be isolated and a selection of the tower designs ordered in terms of performance. Figure 7.6 shows a some designs proposed by the architect manually and how they compared to several designs generated by Embryo in terms of two measures: building height and solar exposure. Instead of being used to find the *optimal* result, such visualisations act as nudges (Thaler & Sunstein, 2008) towards pursuing efficient tower forms, without forcing a particular direction onto the design team.

Following this initial generation, a smaller selection of towers were to be selected by AG5 architects for further manual development. Unfortunately at this time the live project went on hold, however I subsequently asked AG5 if they could attempt to take forward some of the tower models anyway as a hypothetical exercise. I was interested to see if the generated parametric definitions could be understood and used after

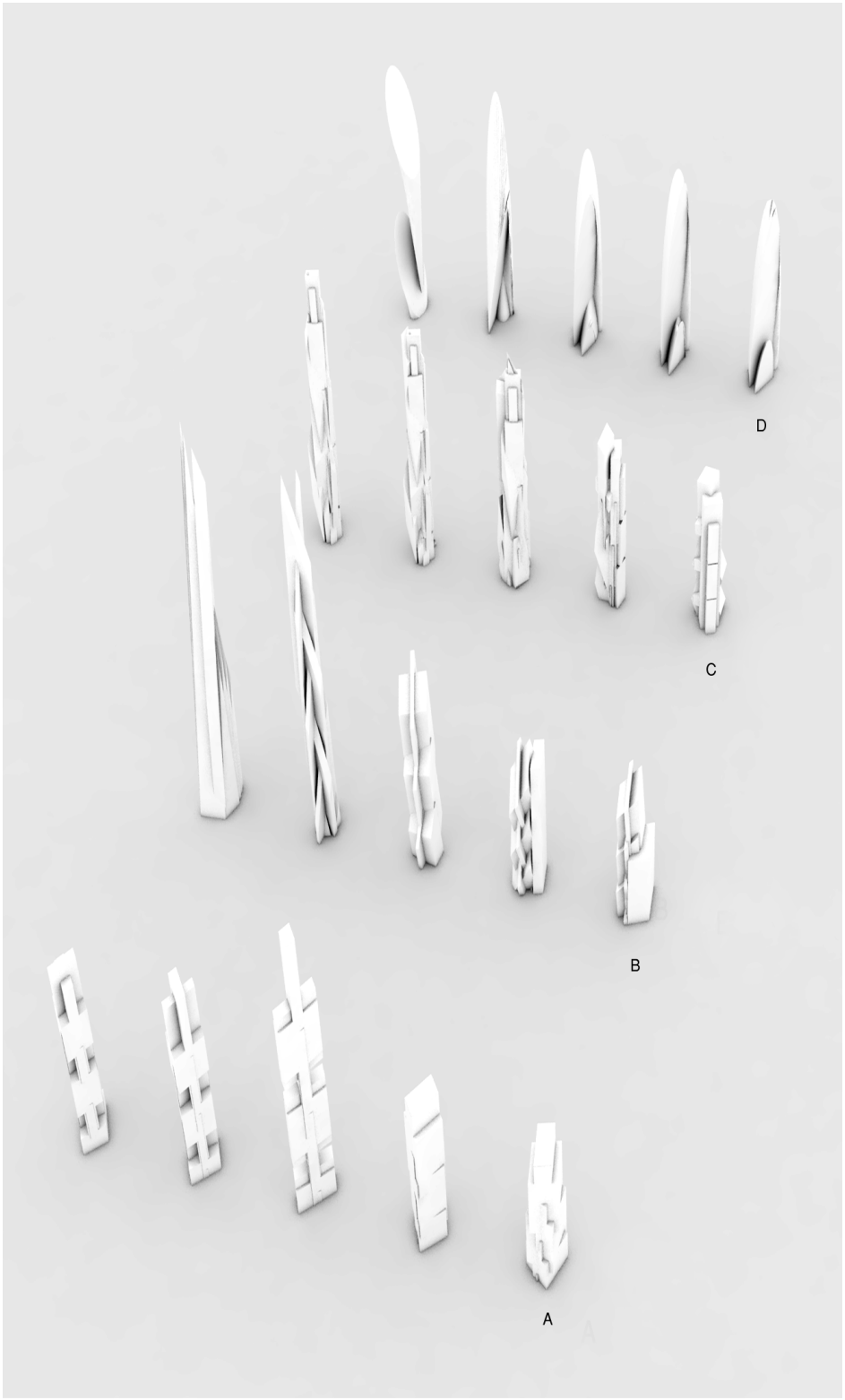


Figure 7.3: Tower phenotypes A to D

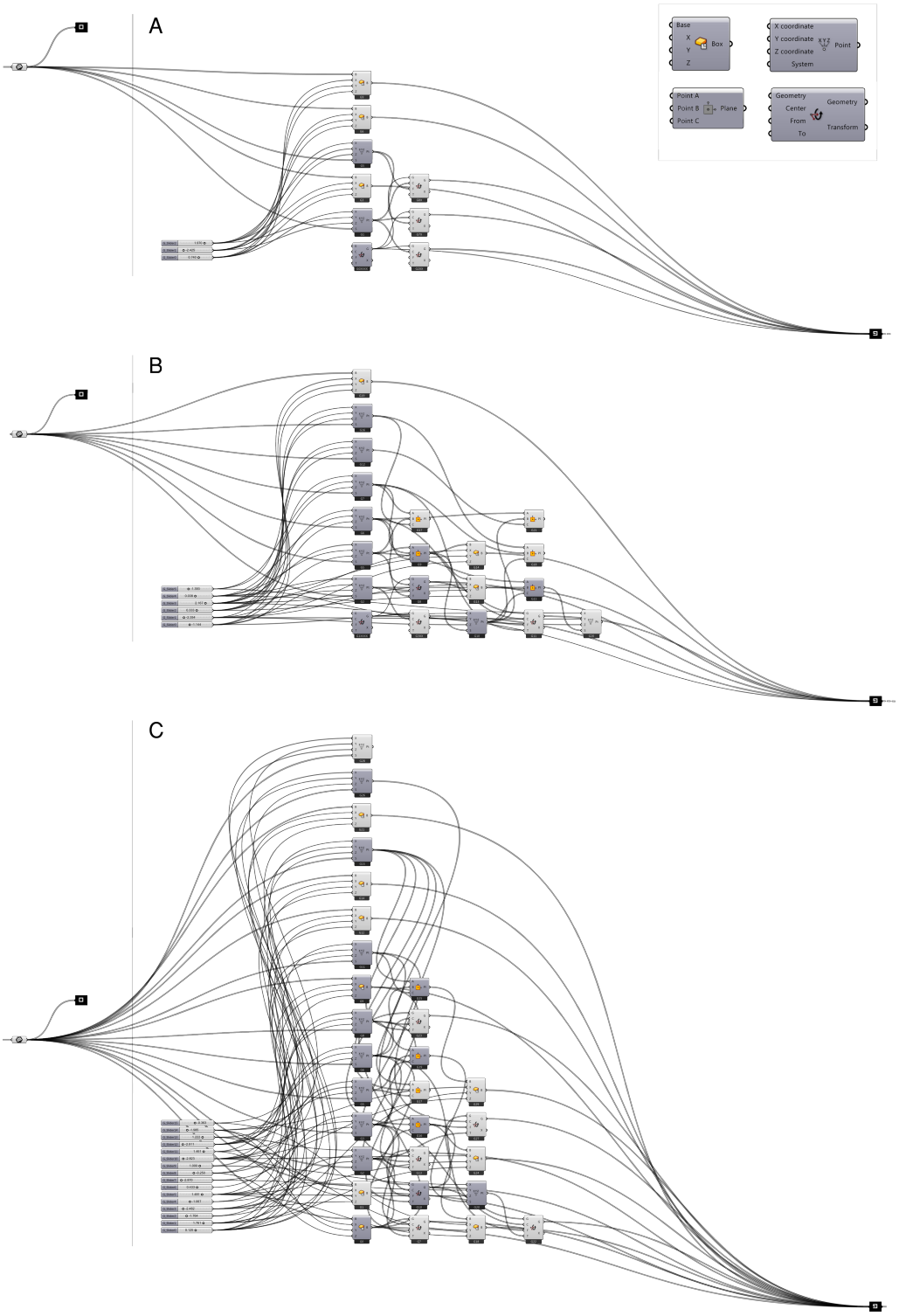


Figure 7.4: Tower graph definitions A to C

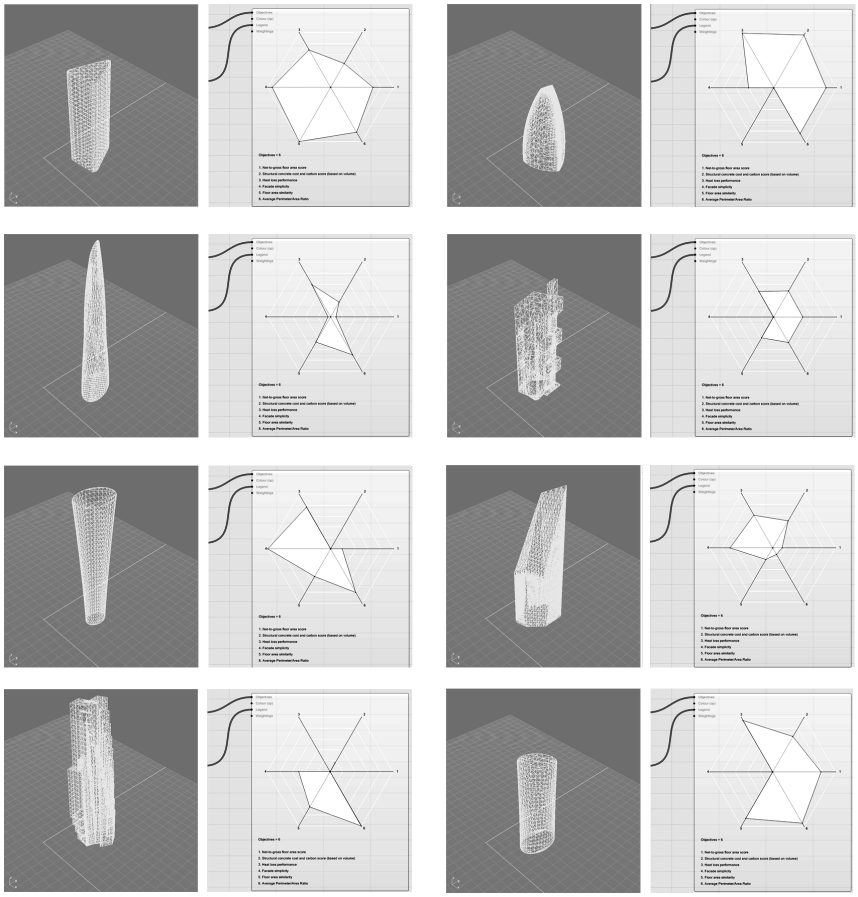


Figure 7.5: Different tower designs with identical gross external floor areas are assesses for multiple performance objectives.

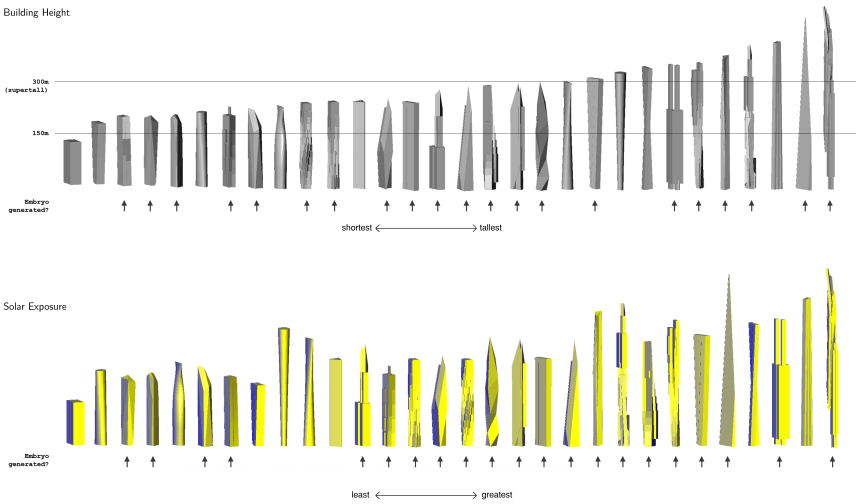


Figure 7.6: Eight architect designed towers with 22 Embryo designs compared. Solar exposure is taken at Summer Solstice throughout the day

being created by a machine, and hence attempt to justify my earlier decision to manipulate parametric models with an explicit embryogeny (see Section 5.5.3 for the discussion). I contacted Daniel Nielsen of AG5 architects and Mark Pniewski at Ramboll to see if they could progress some of the Embryo generated models. Daniel and Mark had both worked on the project since its inception and had a good working knowledge of Grasshopper.

7.1.5. Model Cognition

Nielsen and Pniewski were given the four parametric models examples from Section 7.1.3 and asked to answer the following questions:

1. By adjusting the generated numeric parameters, do you understand their influence or is the experience essentially meaningless?

2. Is it possible to understand how the model is built up hierarchically by looking at the graph or do you just see 'spaghetti?'

3. Can you use the graphs to add additional features post-generation?

Q1:

Nielsen and Pniewski could understand how parameters varied the model for graphs A and B and felt in control of the model, however both struggled to understand the consequence of adjusting the numeric parameters for graphs C & D. As Nielsen commented following the study: "as complexity increases, it becomes difficult to understand what the slider does", although for all examples they could potentially be understood over time: "If you are used to work with Grasshopper it gets more understandable after a while when looking at the relations." Pniewski particularly struggled with the graph for Tower C. For example, when adjusting parameters additional geometry would just appear and disappear unexpectedly. This effect relates to the preservation of 'junk' components (p.134), that whilst useful for evolvability may cause issues when manually adjusting models.

Q2:

In terms of the legibility of the graphs themselves, again parametric model 'C' became the limit of legibility. Both mentioned however that in time all of the definitions be understood, but Pniewski felt "daunted" by the prospect. Disappointingly, the Embryo cognition tools that step through the graph did little to enable the understanding, because although stepping through the dependency structure they gave no insight

into the relative associations when doing so. Stepping through the hierarchy dependencies from each parameter could be a better method in this regard.

Nielsen commented that "...where the [graph] shape is little bit complex it is always difficult to understand graphs other people have done. I think it is a common issue even though visual programming should make it easier for people to understand the relations." Organising the graphs so that numeric parameters have logical names (relating to their inputs) was suggested by Nielsen, with Pniewski suggesting it would be useful for them to be closer to the first input in the hierarchy (as opposed to them being lined up on the left hand side of the canvas).

There was also an issue with the component choices themselves. Nielsen stated that "in the case where the graphs/geometry is little bit complex, it is a challenge to understand *why* something is rotated or *why* there is used a 3pt circle". Here the question was why a particular geometric primitive component was being used at all and how to form a relationship to the chosen functions: "Basically I think the challenge is related to the choice of geometrical generation."

Q3:

When asked to use the models post-generation, Nielsen commented on the advantage of having the model within a parametric environment where there was already a wide range of analysis tools available: "I like to have stuff parametrically, it is easy to put parametric energy, structural optimizations or whatever kind of parametric optimizations on top of the geometry." Pniewski managed to take the simpler graph for model 'B' forward, firstly by deleting some components and then adding new ones to define new geometry but still based on the original rotated angular forms and GEA constraint (fig. 7.7). Feedback from the analysis criteria within Grasshopper helped guide the modifications, something *not* present when using Embryo to initially generate the parametric definitions.

7.1.6. General Observations

During the collaboration with AG5, Embryo made it possible to generate many different tower typologies in a minimal amount of time, and include in a comparison to some of the architect's own designs. In a single design team meeting we were able to investigate multiple performance criteria for various building typologies. This also included completely impossible designs to construct, but they all helped to investigate the solution space for the problem. This small experiment showed the

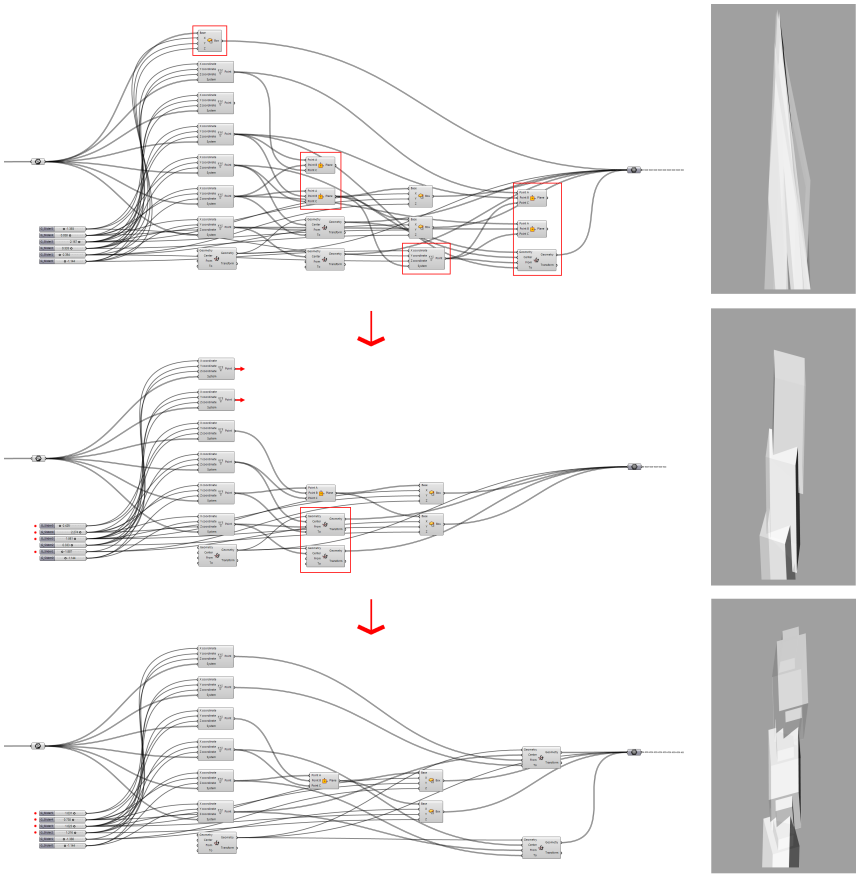


Figure 7.7: Pniewski modifications to design 'B'

potential of using a Meta-Parametric approach, however there were difficulties in understanding the graphs that were generated as we saw in Section 7.1.5. Some general observations were the following:

1. Embryo could quickly generate thousands of designs to explore the design space and thus give an idea of the multitude of possibilities within the set constraints. Even if not used as final designs, these could be used to provide a relative comparison to designs arrived at using traditional methods.

2. At times, the solution process wasn't as fast as expected. Embryo often got held up computing complicated geometry instead of moving onto the next design. This was especially the case with computationally intensive functions such as boolean operations. Bloat, a common issue with genetic programming (Poli *et al.*, 2008) occurred several times and crashed Grasshopper. A possible fix for this issue would be to discount solutions that are taking a long time to solve and simply move onto the next design, although this may preference a certain set of solutions.

3. As Brian Sheldon at AG5 architects later commented, "The process will be even more useful when looking at relations between buildings." (pers.comm.22/03/2013). Although the radar plots gave an

idea of relative building performance for multiple objectives they were limited to a small number of designs with visual comparison then required. Relative comparison of multi-objective criteria between a larger set of designs could be made by mapping options associatively to a lower dimensional space (for visual comparison) - for example using a self-organising map (Kohonen, 2001) or principal component analysis (Hanna, 2007).

4. The process was sequential. A single collection of designs were generated and *then* evaluated instead of an interactive or evolutionary approach. We saw that Pniewski would modify the generated graphs being guided by the analysis, but this was after Embryo had finished. A more interactive method may have been more appropriate in terms of guiding the process iteratively through the paradox of choice (Schwartz & Ward, 2004), engaging in a wide design exploration with quantitative and qualitative objectives that changed over time. We will see in the next experiment (Section 7.2) how this experience was improved.

5. The cognition result was slightly disappointing. There became a limit on the complexity of the graph and the connection to the human design team. For this study, a limit of approximately 20 components seemed to be the maximum.

For the last point with regards cognition, the following improvements are proposed.

7.1.7. Improvements

1. Make the graphs more legible:

In light of the suggestions by Nielsen and Pniewski, a manual clean of graph D was made post-generation (fig. 7.8) with the result being more legible for them both. For instance, locating parameters closer to inputs (not lined up on the left hand side) and removing terminal components that were not used for anything (such as vectors) seem to be good advancements in generating legible graphs. Better methods to untangle graphs post-creation must be considered, or else integrated when graphs are being constructed.

2. Use components that embed a particular heuristic developed by the design team (as opposed to any-old geometry):

The use of geometric primitives for no reason left Pniewski and Nielsen distanced from the graphs. As well as setting up constraints and objectives in early design team meetings, some thought to the components

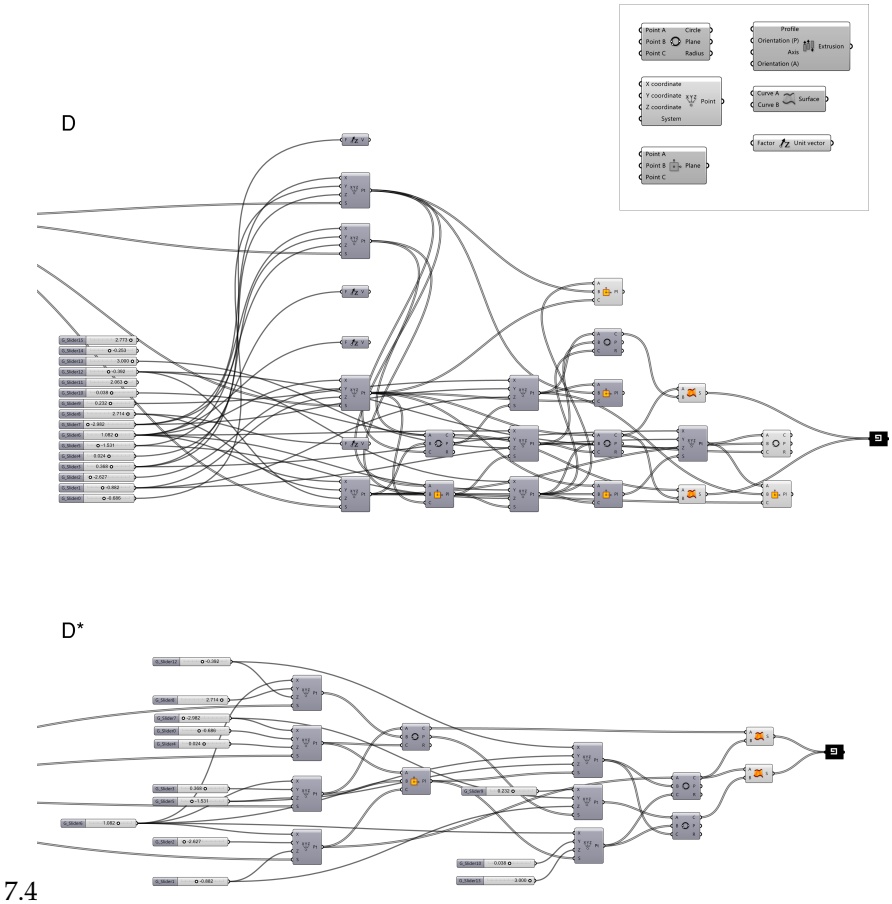


Figure 7.8: Manually untangling the Option D graph and removal of superfluous parameters and components

7.4

chosen and why would help with this. We will see in the next example with 3DReid that building up a set of components (or rules) that embed heuristics developed by the design team offer a better connection at this level and therefore help when attempting to understand the final definition (refer to the earlier discussion on this in Sections 5.4.2 & 5.4.3).

3. Build up complex graphs *over time* instead of being presented with one immediately:

If graphs of higher complexity are to be generated with Embryo, this needs to happen progressively in order to form a narrative (not simply Embryo presents the human with the graph and wishes him/her good luck!). Again, one possible option could be using an evolutionary method with artificial selection in order to develop graphs of higher complexity over time whilst retaining a connection to the design team. Regardless of complexity, a more interactive experience during graph generation would help the design team use Embryo to better understand the performane criteria and their relative weightings.

7.2. Experiment 2: 3DReid Architects

A second chance to test Embryo arose in late 2014 for a high density residential project in Tower Hamlets, London with 3DReid Architects [33Tow]. The project was at the concept design stage and therefore another good opportunity for assessment. The site is a 20,000m^2 flat plot of land just north of the Thames with a series of medium to large sized buildings surrounding the site. A series of design team meetings (which included some Embryo training) were held from December 2014 - January 2015. These meetings resulted in the development of a concept scheme that was developed from a Grasshopper initially generated by Embryo.

7.2.1. Set up

As opposed to randomly producing a series of models that were then tested post-creation (as with the previous experiment) in this case the performance of each model was updated in real-time. This brought the experience closer to the software developed in Section 5.3 for the ENI Headquarters competition [20Eni], but with the added benefits of a parametric definition. There were two quantitative aspects explored through the models:

1. A target number of residential units (transformed into a target *gross* external floor area of approx. 100,000m^2)

2. Proportion of residential units with an unobstructed view towards Central London (classed as St.Paul's Cathedral)

As well as these identified quantitative objectives, the design team were free to qualitatively assess the designs that were produced using Embryo. In short, we were looking for designs that were mixed density, produced well-lit courtyard spaces and had high spatial connectivity at ground level (no cul-de-sacs). Although one could potentially conduct spatial analysis Hillier & Hanson (1984) for the latter, it was thought that manual observation and judgement were adequate.

The set up enabled the design team to assess the performance criteria in real-time in Grasshopper. During the initial search, we were also aware that the model could potentially be adjusted (using the numeric parameters) post-generation to meet the requirements. The set up in Grasshopper can be found at Appendix H.1.

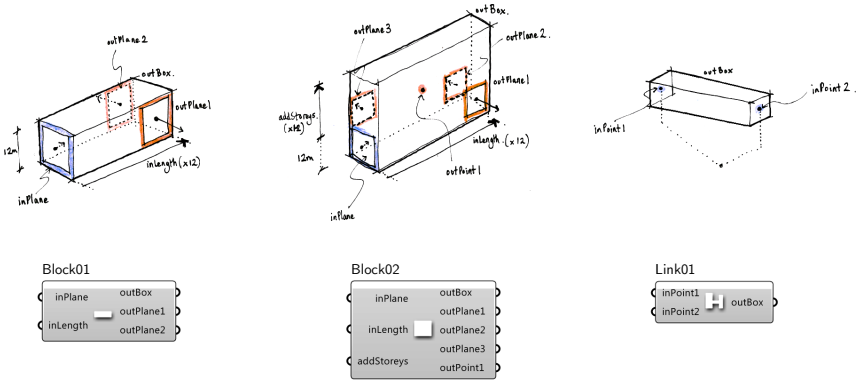


Figure 7.9: Grammar rules specific to the project made into Grasshopper components

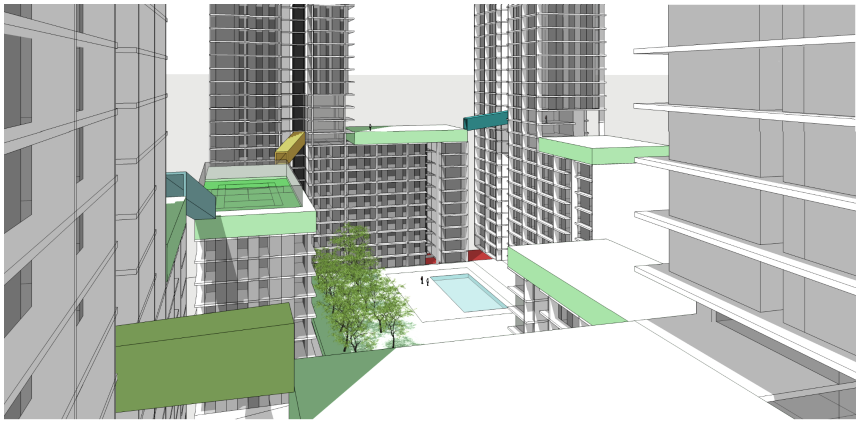


Figure 7.10: Concept design option for a previous project: Putra Heights, Malaysia (image: 3DReid Architects)

7.2.2. Modelling

The architect was interested in being able to model different orthogonal designs using a computational approach that incorporated apartments of a specific size. In response to the issues with Gran Rubina Tower, this time we worked together with the architect to set up a series of apartment block shape grammars that would incorporate this logic and create different combinatorial options within a parametric modelling environment. The rules manifested as Grasshopper components (fig. 7.9) with heuristics both *specific* for this project and based on the experience of 3DReid. This included two types of residential block: one with a set minimum height of 12m reflecting 4 storeys and one with a variable height. An additional third component formed a link bridge between residential blocks and contrast the orthogonality of the block layout. This was a socially driven heuristic based on previous work by the architect (fig. 7.10), providing connection to isolated residential cliques inspired by small world networks (Watts & Strogatz, 1998).

The three shapes each had input and output planes where connections were made, essentially setting up the growth rules for the system. It was simple to set up these rules within a parametric modelling environment like Grasshopper that allowed custom components to be made and

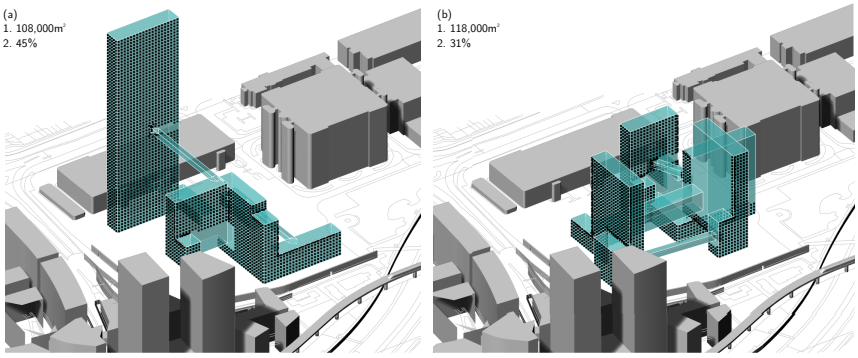


Figure 7.11: Two example designs with the two performance criteria shown. Apartments with unobstructed views to St.Pauls are indicated with a dot

developed quickly - indeed this ease of adaptability at the level of establishing the rules is important to avoid a generic 'logic of architecture' to creep in to such systems (see Section 5.4.1 for further discussion on this). The generated graphs used starting seed planes to *grow* designs, set as tagged parameter outputs from the parent model. Once created, post production work included adding some notional columns to support the structure if it was raised from ground level.

7.2.3. Design Exploration

During design team meetings with the architect, the use of Embryo was relatively undirected. We found ourselves first playing with the random override and then making minor adjustments to models using the directly encoded genes (see Section 6.3.5). We could quickly alter the probability of generating models with less or more blocks of certain types by altering their number in the ingredient component section of the Grasshopper canvas. At times we explored using a metaheuristic (Galapagos) to generate schemes with a high percentage of 'rooms with a view', but for this project we soon found that this had a direct correlation to very high towers which were overruled by knowledge of local planning conditions in the area. Instead, we relied much more on the real-time analysis to *guide* our search alongside viewing the scheme in model view (fig. 7.11). Although only massing models were generated, it was possible to get a feel for the space and how each design worked as an overall scheme.

A selection of some of the designs progressed of varying complexity can be found in Appendix H.2. One such example is shown in fig. 7.12, showing both the graph and associated phenotype.

Designs could be generated that were not contained within the site, and generated parameters then adjusted post-creation whilst maintaining associative relationships between components. An example is given in fig. 7.13, whereby Option 'C' is adjusted to meet the site boundary

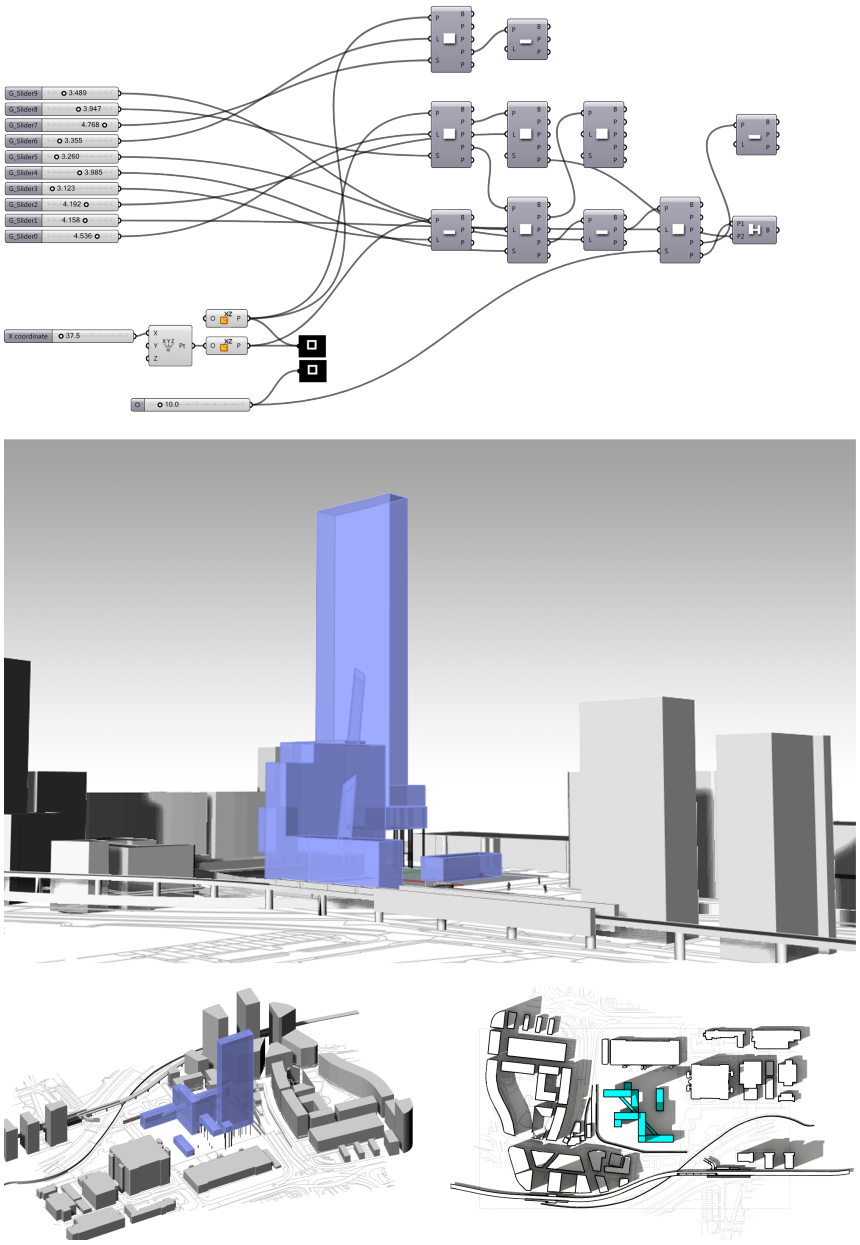


Figure 7.12: Graph and phenotype for Design 'A'

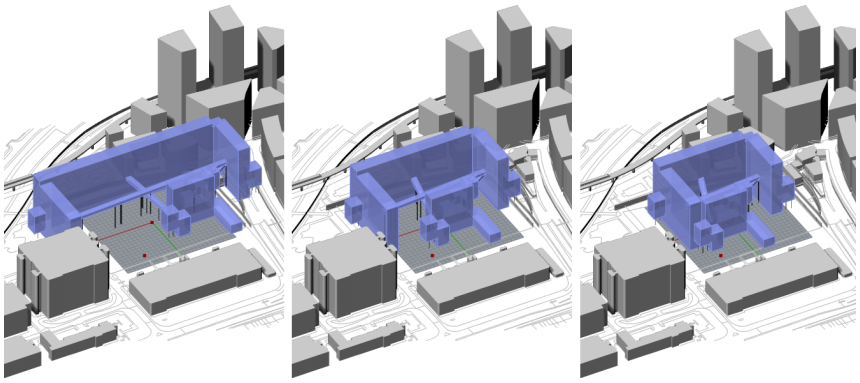


Figure 7.13: Manually adjusting Option 'D' post creation

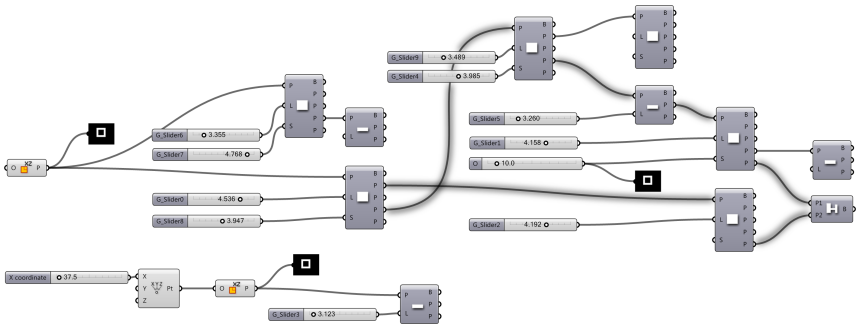


Figure 7.14: Graph from Design 'A' manually untangled. The diamond in the DAG is highlighted

constraint and get to achieve the desired approximate volume to meet the residential unit number objective.

The link bridges form circulation pathways between residential blocks. As components, these required two 'Point' data type inputs supplied by the residential blocks, resulting in 'diamonds' in the DAG and a joint dependency structure not found with a tree (see fig. 7.14 and previous discussion in Section 6.2.4).

7.2.4. Graph Complexity as an Objective

It was found that generating simpler graph definitions was a factor in the design search but without compromising the nature of the generated designs. As with the previous experiment on Gran Rubina, there was an approximate correlation to the complexity of the graph and the complexity of the associated phenotype due to the explicit nature of Grasshopper. Indeed, Charlie Whitaker of 3DReid later commented that he was "actively looking for simpler graph [definitions] whilst maintaining interesting designs of sufficient complexity" (pers.comm., 21/01/2015).

This meant arriving at definitions with around 15-20 components (Design 'C' for example, as opposed to Design 'F' which was too complex). Although reducing components was preferred, this wasn't necessarily the case for the numeric parameters. As Whitaker commented, "After a

while, I had the thought that the more child sliders the better; provides a greater choice of starting values and seems to generate richer models."

Issues arose when one parameter could control completely different function inputs that were part of disparate dependence structures through the graph. In such cases, it becomes hard to have any meaningful connection when making manual adjustments after generation. As Whitaker commented for such definitions, "I think most of my time has been spent in a 'tweak slider and observe' behavioural pattern."

Again, it was interesting that *increasing* the number of variables whilst increasing the size of the graph did give a better feeling of control. The 'one-to-one' setting in Embryo (see Section 6.3.4) was used to good effect in this regard, and one can see for designs A to F how one numeric slider is associated with one component input in the graphs generated (note: this does not generate trees, because components generally have more than one input and output). In reality there is a balance to be struck between having too few metric parameters to understand their influence and too many to keep track of. As Davis (2013) states: "An ideal parametric model would encompass all the variations the designer wants to explore within the smallest dimensionality possible."

7.2.5. Post-Generation

As with Gran Rubina, we attempted to take one of their preferred Embryo designs forward and use it as a regular Grasshopper definition. Charlie Whitaker of 3DReid selected one the models generated using Embryo that he thought had potential to be developed. This time, instead of building on a parametric model made by another consultant, this was one made with the help of a machine. Whitaker made the following amendments to the model (in order):

(a) Understand an initial definition and model by Embryo

(b) New building and three link bridges added

(c) Additional long building added to the north and parameter (*) adjusted (to constrain design to site)

(d) Generated building made taller (to meet GEA objective) and moved above ground level (to increase circulation to south)

(e) Entrance space created (to open courtyard space)

(f) Graph clean and post-production work (façade patterning)

(g) Final analysis against quantitative criteria

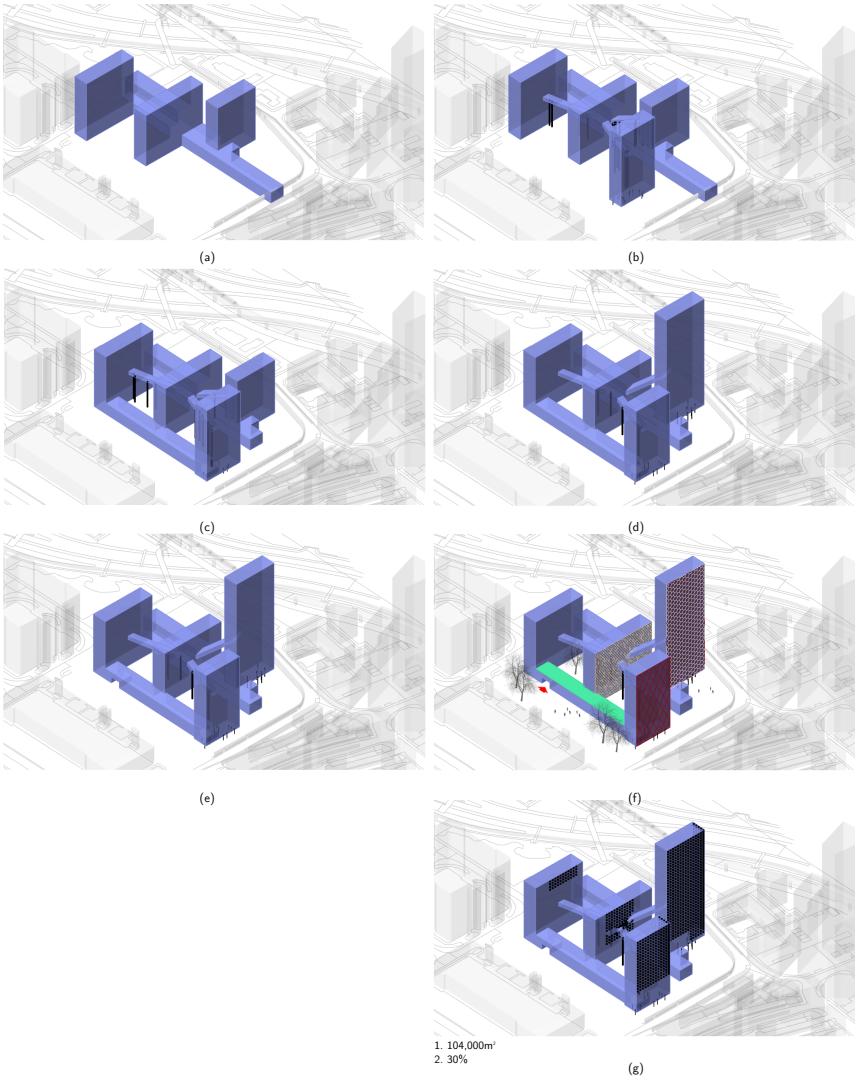


Figure 7.15: Taking an Embryo design forward

The alterations to the model can be seen in fig. 7.15 with the associated graph change from (a) to (e) shown in fig. 7.16 (the full process can be found in Appendix H.3). During the modifications, the design was assessed against the performance objectives, with the final results shown in (g).

The result was successful, at least for this relatively simple definition. Whitaker was able to use the parametric definition to add additional aspects to the model and adjust sliders in order to meet the site constraints. For the latter, we can see the advantage of the parametric definition that makes such adjustments possible whilst maintaining the definition structure. Whitaker later commented that he felt he had joint-ownership of the model due to an understanding how it worked. Although the example is a simple one, it shows the potential of keeping graphs legible if generated (at least in part) by a machine.

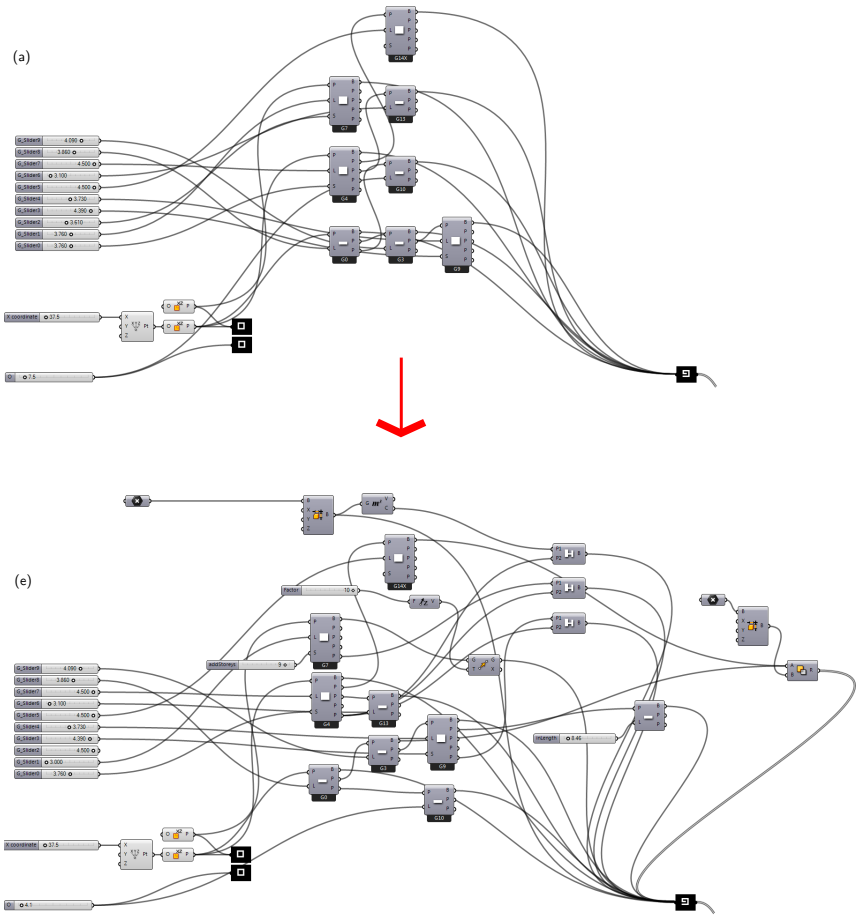


Figure 7.16: Manual developments to an Embryo model by 3DReid Architects

7.3. Experiment 3: Shape Analysis

In the previous two examples Embryo was used to generate multiple parametric models as part of a design exploration exercise. In this experiment, I generate parametric models and combine them with a metaheuristic solver in order find a given single objective - a target geometry. The combinatorial problem of generating a parametric model definition that will itself generate a target dumb CAD geometry is an interesting challenge as it offers the possibility of generated new parametric models in place of existing ones for the same geometry, thus offering new possibilities for design development outside of existing parametrisations (see the earlier discussion on page 105).

An initial test implementation was conducted to show the capability of Embryo to generate parametric models from scratch that generate a target geometry. This geometry was presented to Embryo, with a shape analysis conducted as a performance criteria. Shape analysis is the study of shape matching and recognition of geometry (Costa & Cesar Jr, 2000). The first test used a simple target geometry of two forms, one on a box on a twisted plane, and one a trapezoid aligned with the WorldXY plane.

The shape analysis I used in the study was a simple point cloud comparison summarised as follows:

1. For each vertex in the generated model, find closest vertex in the target geometry and record the distance.
2. For each vertex in the target geometry, find the closest vertex in the generated model and record the distance.
3. Sum all recorded distances.

The task then became to minimise the distance total between closest vertices. The parameters used by Embryo to attempt to solve the problem were the following:

- 20 Sliders.
- 20 Components.
- Three input components: Point by XYZ, Plane by 3 points, Centre Box by Plane & XYZ.

7.3.1. A Suitable Metaheuristic

Even for such a simple set up, the number of possible parametric models is vast, not even taking into account the infinite possibilities for the slider

values. Assuming only a single connection per component input, there are $((12+8)! - 8!) \cdot 3^{12} = 10^{24}$ possibilities, more than all the grains of sand in all the world's beaches combined! Clearly a brute force search process is not suitable for such a large design space.

Simulated Annealing was therefore selected as a suitable metaheuristic (see discussion Section 2.3.2), namely the *Galapagos Component* (Build 0.2.0448). At the time of writing, this solver is included with Grasshopper and hence can be incorporated into a parametric model schema very easily. As the author of Grasshopper and Galapagos David Rutten comments, (pers. comm., 20th December 2014), simulated annealing in Galapagos is a good candidate for finding many promising optima, while simulated evolution is much better at refining a single solution.

The simulated annealing solver was set up with a cooling rate of 0.97 and a drift rate of 0.55 (the drift rate controls the odds are that more than one variable is changed in Grasshopper).

7.3.2. Results

The results of the simple initial experiment were encouraging, with the SA solver able to find a parametric model that got very close to the target geometry in approximately 200 iterations (fig. 7.17). It became clear that a perfect match was impossible due to the limitations of the input components being just boxes (and not being able to match the trapezoid). However, setting the ingredient component geometry to just boxes showed how that a more primitive set of components could still evolve closely to a target geometry that included more complicated shapes. The parametric model generator did the best it could with the components at its disposal.

Figure 7.18 shows the resulting grasshopper definition following the manual removal of junk components. Firstly, we can see that the first box is simply generated from three number parameters as the target geometry was based on the XYPlane. Secondly, we can see how three points have been made that define a new plane in $\mathbb{R}^3$, which then defines the second box. It is interesting to note that some numeric parameters are shared, i.e. "G_Slider6" is used to set out both a point used for the plane and for the first box.

7.3.3. Increasing Complexity

A second experiment was conducted to test the whether Embryo could use a 'divide' component that subdivides a line into a series of points.

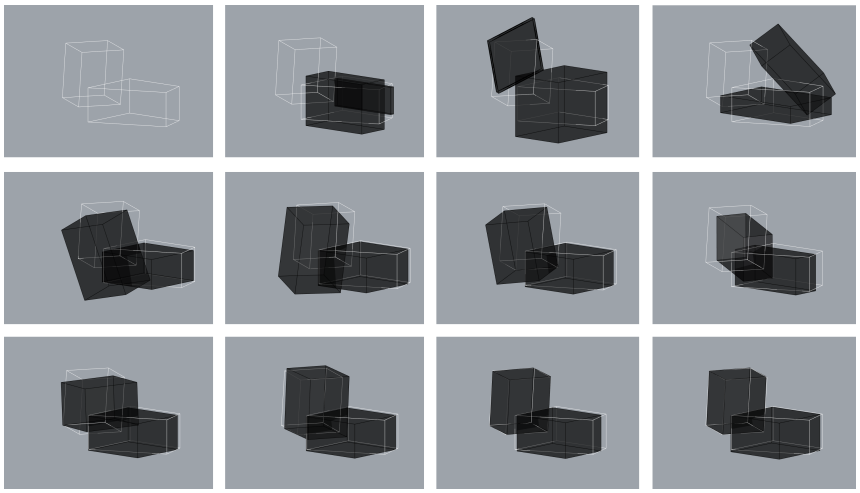


Figure 7.17: Shape matching history using Embryo with the Galapagos Simulated Annealing solver. The target shape is top left.

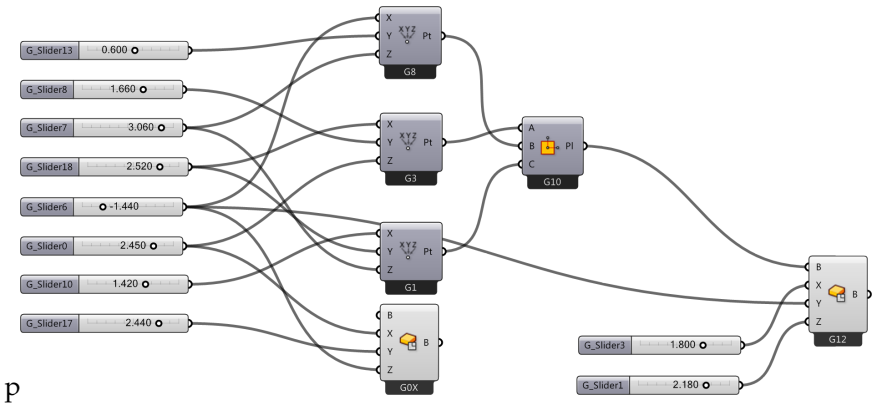


Figure 7.18: The generated graph for the first problem following a manual clean.

p

A more complicated target geometry with a tapering staircase and a separate block was used. The challenge was to see whether a parametric definition could be found that acknowledged the repeatable pattern of the staircase. The parameters used by Embryo to attempt to solve the problem were the following:

- 16 Sliders.
- 32 Components.
- Four input components: Point by xyz, Line by 2 points, Box by 2 points, Divide curve.

The simulated annealing solver was set up with a cooling rate of 0.93 and a drift rate of 0.70. The faster cooling rate was selected so that many runs could be made with the annealing solver, as it was thought more local optima would be present in the solution space. It took several attempts for the solver to find something close to a global optima with the results of one of the successful annealing runs is shown in fig. 7.19. We can see from the top view that the solver managed to generate the tapering staircase quite well.

The generated definition for this solution following a manual clean is shown in fig. 7.20. The first box is generated using two points. The second box has a forth-order dependency rank, made from two points, a line and a divide component (labelled G17). We can also see how this is a DAG - for example the numeric parameter "G_Slider10" being used as an x-coordinate for a point and also to set the number of divisions for the divide component. The parameter controls two distinctly different things which on the one hand reduces the amount of elements on the graph (DAG is more efficient than a tree), but on the other leads to more difficult understanding of the graph and usability - something discovered on both the Gran Rubina and the Tower Hamlets project. Legibility in this sense is not simply measure in terms of the amount of nodes in edges in the graph, but rather the effect the numeric parameters have on its subsequent variation.

Although these are relatively simple examples, the power of being able to generate a model automatically has repercussions that have not yet been explored. One could even use Occam's razor to generate the simplest model for a given geometry by incorporating graph complexity into the search. This could potentially clean some of the spaghetti of overly verbose parametric definitions that could have a simpler representation, and therefore enable better collaboration *between* stakeholders.

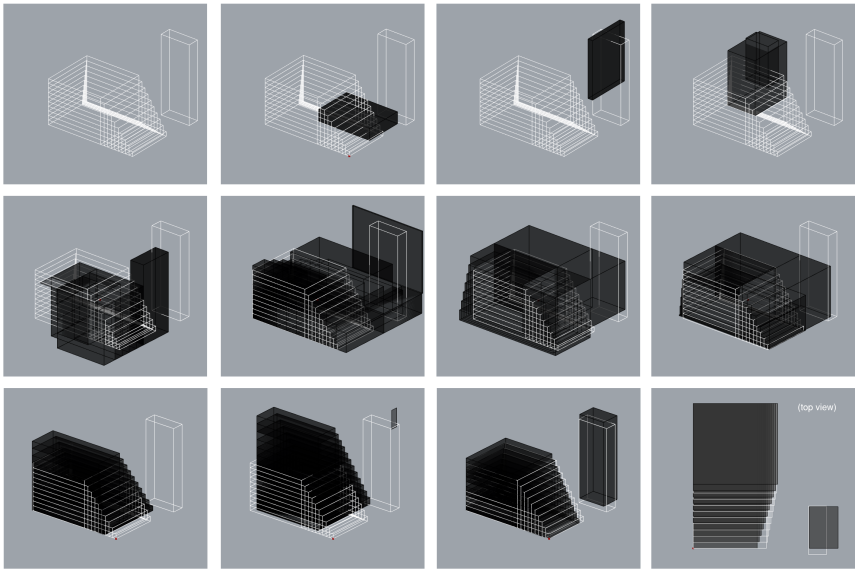


Figure 7.19: A more complicated example involving a series component.

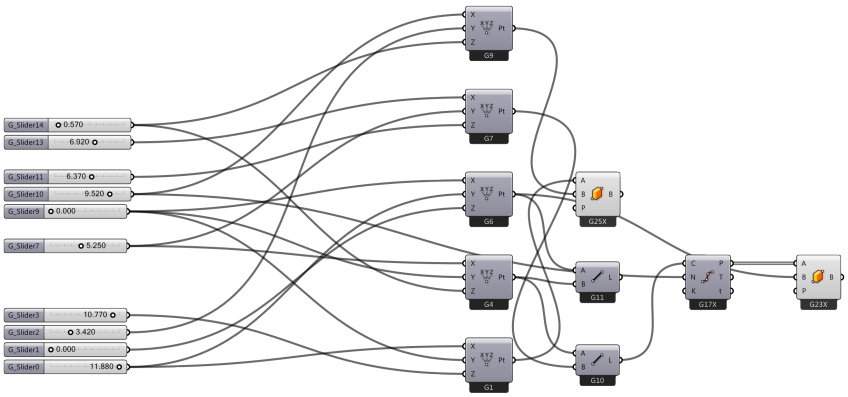


Figure 7.20: The final generated graph for the second problem following a manual clean.

7.3.4. Opening Dead-ends

In Section 4.5.3 it was found that the topological inflexibility of parametric models meant that design teams gradually become *locked-in*, with any large changes to the requirements brief or constraints resulting in the time-consuming rebuild of new definitions. By using a meta-parametric approach to search for definitions, new parametric schemas for individual designs could in theory be formed and thus open up new possibilities for parameter variation. A natural analogy to convergent evolution was given, showing that although two forms may be identical at a given time, how they got there will affect future development.

An example of generating a new model was conducted by taking the tower model 'B' generated by Embryo in Section 7.1 and attempting to find a new parametric definition using different numeric variables, functions and topological structure. This is similar to the previous shape analysis examples, other than in this case a comparison can be made between two definitions and the space of designs they cover (i.e. within their parameter space).

The originally generated model was made from the following components: Point by xyz, 3d rotate transformation, Centre box by plane & Plane by 3 points. As well as these generated components, the model used existing outputs and post-production work such as site rotation and slicing. The new definition was a reduced set of components: Point by xyz, Plane by 3 points and Box by 2 points.

Figure 7.21 shows a simulated annealing run after approximately 300 iterations. From visual inspection, the results were far from perfect, but a reasonably good match for the phenotype although improvements could be made. For example, the fine detail around the tower corner points has to some extent been lost. Nevertheless, for this example it is possible to evolve a parametric definition of similar complexity to the original but with different components, variables and topological structure (fig. 7.22).

This new definition now can be adjusted within the new parameter space to formulate alternative design options. The result of adjusting the parameters in the new model are shown in fig. 7.23. One can compare these designs to that previously shown for Option B in fig. 7.3, as well as a simpler graph structure (albeit with more numeric parameters). Although this is a simple example, it does show that it is feasible - the question then becomes one of legibility, essentially raising the same questions as the previous experiments with AG5 and 3DReid.

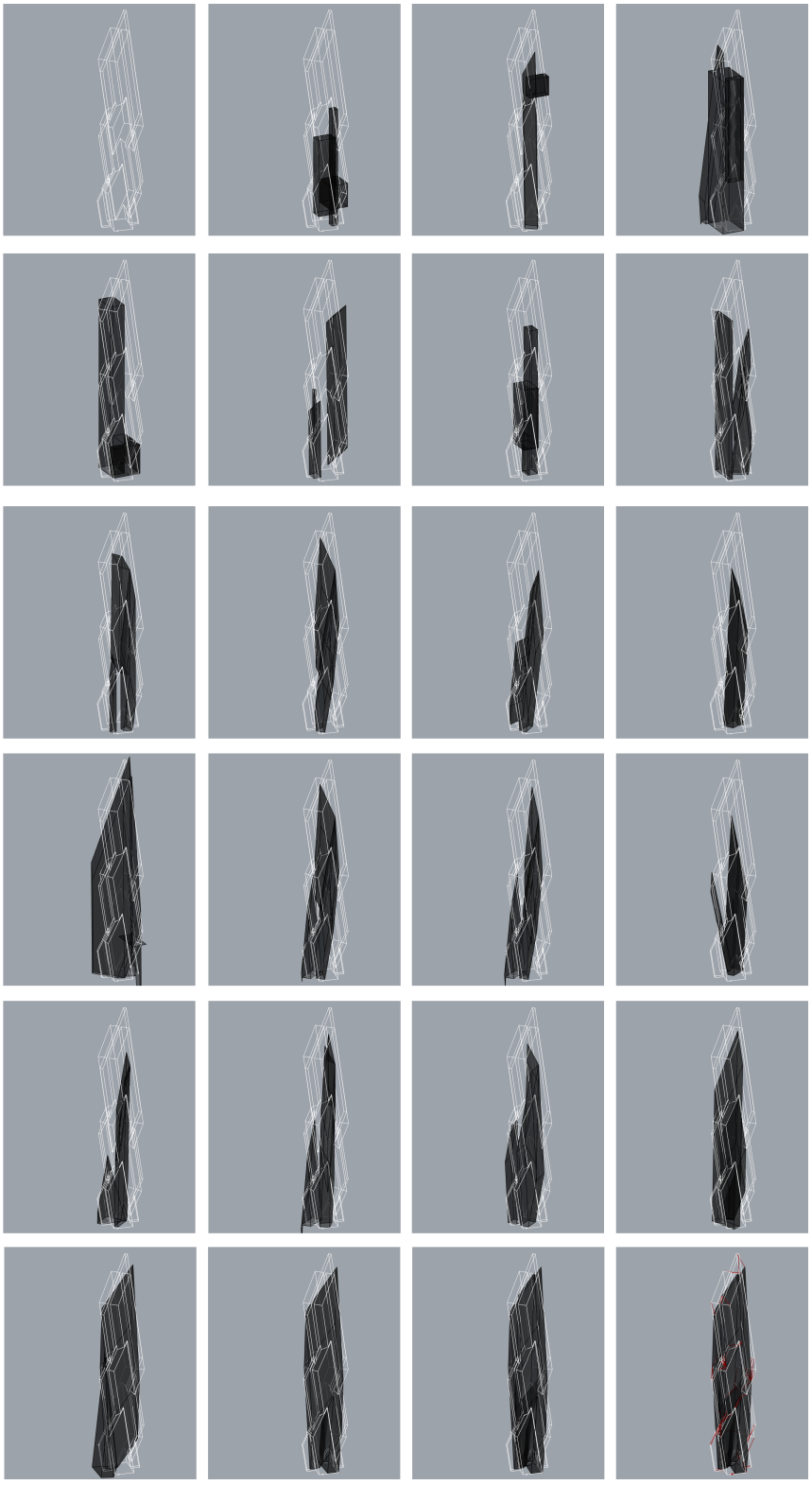


Figure 7.21: Searching for the target geometry from one of the Gran Rubina Option B towers

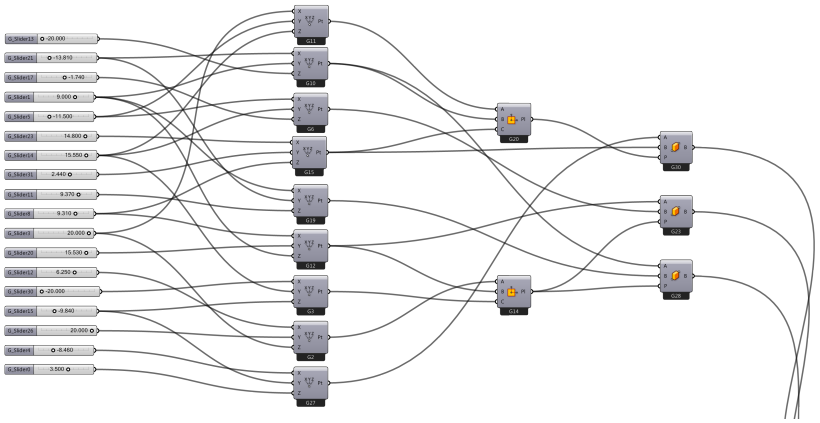


Figure 7.22: A different way of arriving at a similar phenotype for parametric model "B" for Gran Rubina discovered by Embryo.

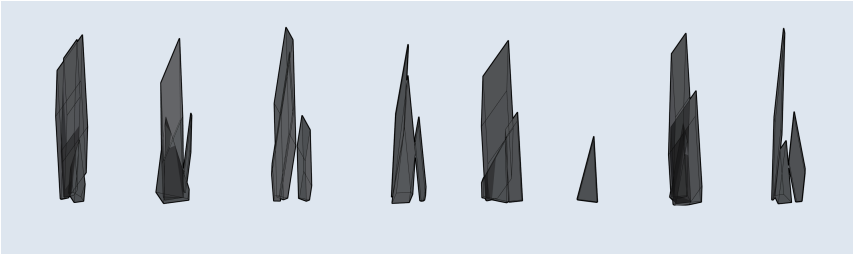


Figure 7.23: Alternative solution space from the new definition

7.3.5. Forget about Legibility

In Section 7.3.4, Embryo was shown to generate a new parametric definition for an existing model. This has been within the constraint of relatively simple graphs to maintain legibility, however if one were to give up on trying to understand the model, is parametric model generation with DAGs useful in its own right? One could argue that being implemented in a environment such as Grasshopper enables the easy use of low-level functions specific to architectural design.

As an example, fig. 7.24 shows a complicated model incorporating the three components developed for the Tower Hamlets project. Using a brute-force search by automatically incrementing the Embryo random override seed, this complicated structure (generated in seconds) was found to get within 1% of a target volume objective set at $4.0 \times 10^6 m^3$. Clearly this graph, whilst *theoretically* possible to understand is practically useless as a cognitive artifact. It does however extend the possibilities of Embryo and Grasshopper as a shape grammar tool in its own right. Here, working at the low level rules and the global objectives become the important areas for the design team with the generated graph being essentially a black box (something I was trying to avoid!).

Even if one cannot understand the graph, we have also seen in terms of evolvability the advantages of an explicit embryogeny (Section 5.5.2).

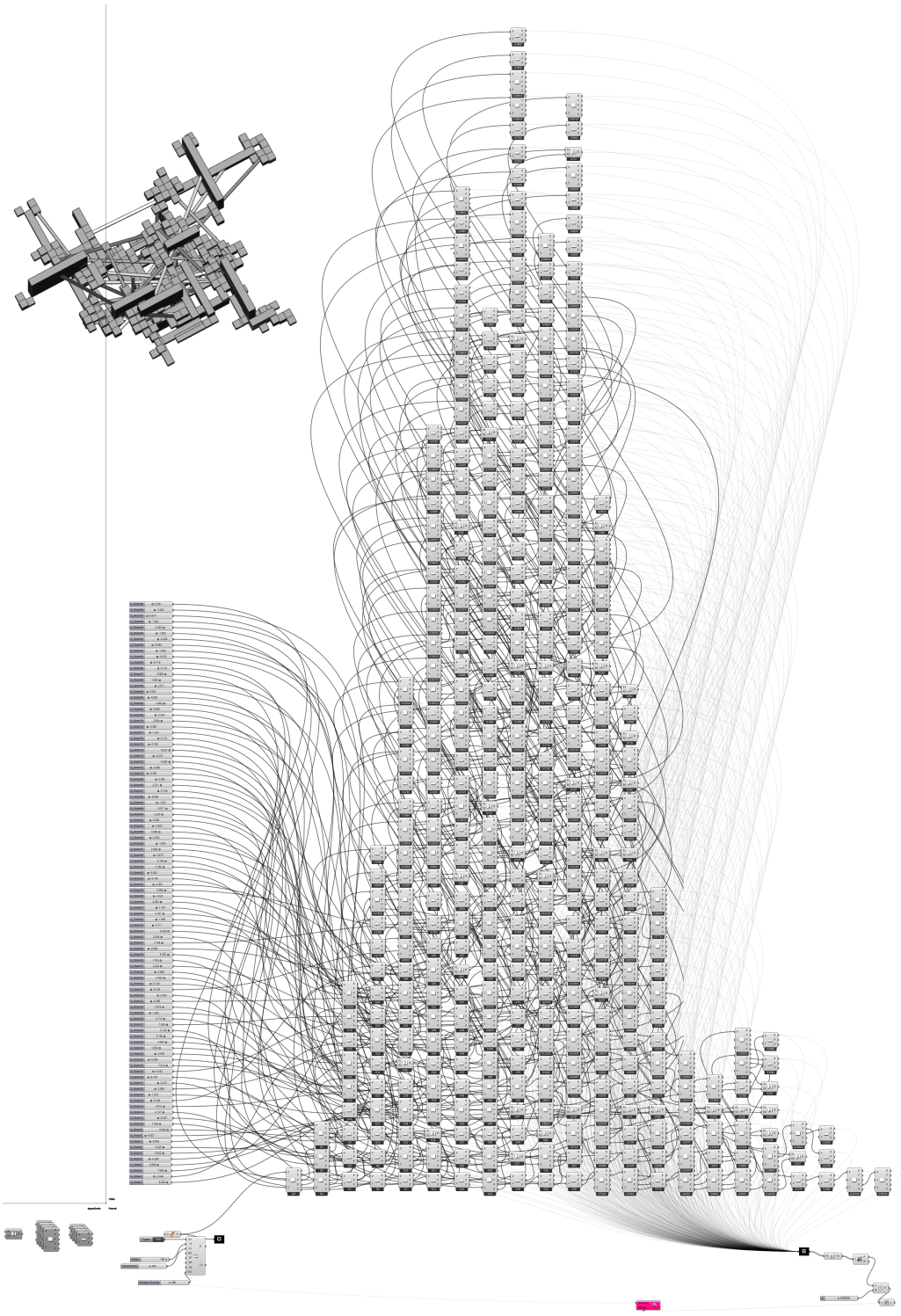


Figure 7.24: An unintelligibl graph and phenotype made Embryo with Galapagos

7.4. Discussion

In this chapter, I have taken inspiration from Genetic Programming in order to generate parametric models automatically, and think topologically in the creation of form. This has resulted in a Meta-Parametric approach, realised in Rhino Grasshopper by writing a custom component called Embryo. Healthy design exploration using parametric models has involved working at a higher level of abstraction, whilst retaining legible graph structures. The 'body-plan' behind the development is now free to evolve with the final result understandable, at least for relatively simple graph definitions. By incorporating a metaheuristic search or simply by randomising the generated results, wide design exploration can take place either to meet a pre-defined geometry or as part of a non-goal orientated design process.

Part V.

Conclusions

8. Conclusions

> "There is a great gift that ignorance has to bring to anything"
>
> Orson Welles, 1960

Computational design is a broad subject, and likewise this thesis has been broad in its initial scope. The research has been a journey both in academia and practice, attempting to understand and clarify concepts in the subject, identify issues inductively and propose and test solutions on real collaborative projects. In the spirit of the Engineering Doctorate, the industrial relevance of the work has played a major part in shaping the academic research and vice-versa.

In Chapters 2 & 3, through the use of case study projects at Ramboll I began to uncover some common themes that dependant on the stage of involvement and nature of each project. In short, the earlier I was involved, the less applicable standard 'problem solving' approaches common to an engineering practice applied. This translated into using parametric modelling tools in Chapter 4, but these were found to be too inflexible for the concept design stage. However, some of the alternatives to parametric design suitable for the complexities of early stage were not suitable in a collaborative environment (Chapter 5), leading to a return to parametric models and the development of Meta-Parametric design (Chapters 6 & 7). This 'trojan horse' approach of manipulating an existing software such as Grasshopper for wide design exploration was final part of this story.

8.1. Meta-Parametric

8.1.1. Summary

In response to Chapters 2-5, this work aimed to combine parametric legibility to multiple stakeholders with an increase in flexibility for the early design stage. Concept design is a complex wicked problem that has largely been avoided by software developers. As Mueller (2011, p.21) states:

> "Intelligent conceptual design tools will support early design as the anticipation of various futures in multiple ways. They will include parametric engines for rapid development of design variants that allow exploration of a larger solution space at a finer granularity, and perhaps encourage parametric extrapolation beyond the pre-design or preconceived solution space"

This is the aim of a Meta-Parametric approach, to avoid taking a path down a design route before understanding the problem whilst still having a strong focus on *process*. Addressing this issue is becoming increasingly important, because of the direction parametric design is taking and its increasing mainstream popularity. In Grasshopper for example, decision support tools are becoming better integrated and are moving earlier and earlier in the design process. In addition, an increasing number of third-party components can offer quantitative feedback on building performance.

These will only increase in the future due to improvements in computing power and software capability. In addition, metaheuristic solvers linked to parametric models are becoming available to the masses. Previously, (including at the start of this thesis) one had to program their own metaheuristic or at least know how to import and implement an appropriate library which required knowledge of programming in itself. However, the 2011 release of Galapagos for Grasshopper was an important moment, because knowledge of coding was no longer necessary for designers wishing to engage in metaheuristic search processes, just an ability to understand a visual programming environment. Solvers for other associative modelling packages such as Autodesk DesignScript and Dynamo for Vasari are surely inevitable in future years.

8.2. Embryo Results

Chapter 6 of this thesis introduced Embryo, a Meta-Parametric plug-in for Rhino Grasshopper developed by the author. In Chapter 7, this was tested on a number of projects in relation to the thesis question set out in Section 1.2.2:

> *How does the use of a Meta-Parametric approach (using Embryo) assist design teams at the concept design stage?*

This was broken down into three sub-questions regarding how designers were using Embryo which were explored on the Gran Rubina Tower [29Grt] and Tower Hamlets [33Tow] projects:

1. The design space of a generated model (variability).

2. The best fit to the objectives (performance).

3. The complexity of the graph (intelligibility).

8.2.1. Model Design Space

In Section 4.5.3 the limits of parametric models were discussed in terms of the design domain traversed by their parameters. How the model is parametrised describes it variability. Following generation, design teams were able to explore this domain by adjusting sliders and thus gaining feedback on the nature of the model.

It was found that understanding the design space was purely a human task, and took time. On Gran Rubina Tower, issues arose when one parameter could control completely different function inputs that were part of disparate dependence structures through the graph. In such cases, it becomes difficult to have any meaningful connection when making manual adjustments after graph generation. It was therefore found that by increasing the number of parameters helped with this understanding, decreasing the amount valency of each parameter. The one-to-one setting on Embryo helped achieved this. There was a balance to be struck between having too few metric parameters to understand their influence and too many to keep track of.

Models that had 'junk' components that would suddenly become active under certain conditions and change the model unexpectedly gave a feeling of loss of control. Essentially, the parameter space was discontinuous. Whether or not this is good or bad is uncertain, but perhaps an additional setting for Embryo could remove the possibility of such models being generated by checking *multiple* parameter values for each design and checking whether the same components remain valid or generate errors for all. Having this as a user control means junk components can still be used during metaheuristic search (see Section 6.3.7).

This limitation also arose in a different context during the experiments. Essentially, the design space for each generated model was not evaluated *during* the process but afterwards. Sampling the nature of the parametric model with a random set of parameters could be an interesting approach. Indeed, Hornby (2003) has already explored this whereby parametrised designs are evaluated multiple times for different input parameters as part of an evolutionary search.

8.2.2. Model Performance

One of the main advantages of using Grasshopper was not just in terms of modelling but also analysis. It was possible to incorporate both quantitative and qualitative analysis, firstly after many parametric models had been generated and reduced to mesh geometry (Gran Rubina), and also during a parametric model search as part of an interactive experience (Tower Hamlets). For the latter, a metaheuristic solver was explored during the search (and then able to be easily discarded).

With Gran Rubina Tower, Embryo presented a range of options to the design team. This left the participants more distant from the parametric models produced, as well as how each performance criteria should be weighted. Tower Hamlets however provided a richer experience because our motivations could change during the model search, and help us to explore the design problem and our relative weightings between objectives. In short, we were discovering what was most important *whilst* using Embryo and not afterwards like on Gran Rubina. This relates well to the earlier discussion in Section 4.3, when addressing the problem of combining [objectives] together into some overall assessment (Lawson, 2006).

For Tower Hamlets, the performance of the model was not limited to the geometry created but also the parametric model definition itself. Simpler parametric DAGs were preferred so as to provide the possibility of understanding them, whilst not becoming too simple as to limit the complexity of the phenotypes explored.

8.2.3. Graph Intelligibility

As mentioned above, as well as the performance objectives, the level of graph complexity became a major factor in designers choosing one design over the other. For both projects, graphs of around 20 components seemed to be the limit, restricting Embryo to simple concept design studies if the definitions are to be understood. Clearly better methods are required for Meta-Parametric working at higher levels of complexity, and some possibilities are put forward in Section 8.3.

As mentioned in 8.2.1, it was found that *increasing* the number of parameters helped to create clearer graphs, decreasing the amount valency of each parameter.

The use of geometric primitives whilst offering multiple interpretation, actually left the designers more distant from the graphs on Gran Rubina (see 150). On Tower Hamlets, the components embedded some of the

design team's experience and hence the components themselves had some meaning, both at the level at the component *and* their use within the generated graphs. This relates to the discussion at the end of Chapter 5 (see Section Discussion), where understanding not just low-level rules but how they are combined explicitly was argued as being useful in a collaborative context.

8.3. Further work

8.3.1. Better Cognition tools

As mentioned in Section 7.1.5, somewhat disappointingly the Embryo cognition tools that step through the graph did little to enable the understanding. This may be because although stepping through the dependency structure, they gave no insight into the relative associations when doing so. Instead, highlighting the hierarchical dependencies from each numeric parameter has much more potential in understanding the graph and its hidden structure. In Section 6.3.8, the 'transactions' approach by Generative Components seems to be a good reference in this regard.

Methods to *untangle the spaghetti* and provide a better layout of numeric parameters as discussed in Section 7.1.6 would also be useful additions.

8.3.2. Recursive & Cyclic Structures

As discussed in Section 6.2.3, recursive structures were used by Coates and Senatore in generating L-System models, using rewriting whilst still maintaining a low degree of epistasis in genotype-phenotype mapping for good evolvability. Sims (1994) also used recursive structures for the creature generation. It would be interesting to explore such an approach in terms of DAG structured parametric models - indeed directed graphs *with* cycles have been manipulated this way in other fields such as artificial neural networks (Boers & Sprinkhuizen-Kuyper, 2001) (fig. 8.1). One could imagine perhaps incorporating cycles in Grasshopper using components such as 'Hoopsnake' (Chatzikonstantinou, 2013) but as part of a recursive graph structure. As well as reducing encoding length, this rewriting a different scales may also help with cognition for example with regards evolving more complex structures with repeated sub-graphs.

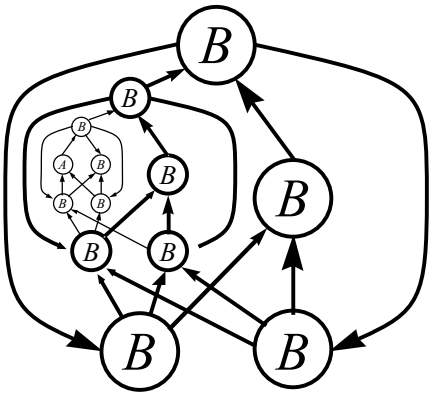


Figure 8.1: An example of a recursive 'Graph-L-System' (Boers & Sprinkhuizen-Kuyper, 2001, p.17)

8.3.3. Hierarchies of Scale

The greatest strength of parametric modelling tools is that they make explicit the geometric process. Their greatest failure however is that graph structures can quickly become illegible to anyone other than the author, fast resembling 'spaghetti' that cannot be easily untangled. Indeed, sometimes the author of his/her own model can struggle! As an example, throughout the case studies in Sections 7.1 and 7.2 it was found that only simple models using primitive components were useful post-generation. For the Tower Hamlets case study, sticking to relatively simple definitions that could be understood became one of the objectives during the initial exploration.

This suggests that if more complex graphs are to be understood and used in any meaningful way, they should be broken down into parts at a series of scales that make up a whole similar to dynamic programming. This could be recursive structures as above, or distinct modular subgraphs each being generated by Embryo and populated as part of a larger system with complexity at different hierarchies of scale (Simon, 1962). Grasshopper already uses a modular structure in the form of 'clusters', so one could imagine Embryo generating graphs within clusters. Again, there is years of development in artificial neural networks to learn from in this regard (Roberts & Turega, 1995).

8.3.4. Artificial Selection

Throughout this thesis, it has been argued that an explicit representation of the development process should be retained so that any evolved parametric models by the machine may be cognised by human stakeholders. However, although touched upon during the Tower Hamlets experiment, how multiple stakeholders can engage in the generation of parametric models in a staged process has not properly been explored. This was suggested at the end of the design exploration experiment (Section 7.1)

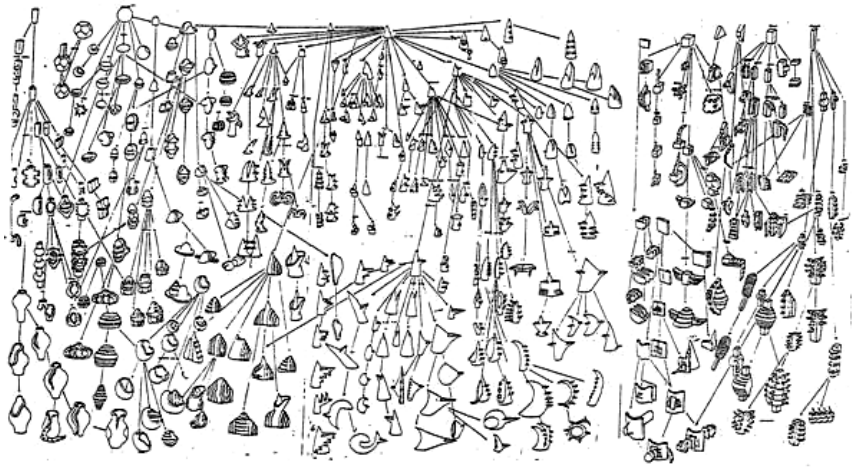


Figure 8.2: William Latham: generative art. Source: (Todd & Latham, 1994)

as a better approach than simply the random generation of many models and the paradox of choice (Schwartz & Ward, 2004) this entails. Using an evolutionary method with artificial selection is one possible alternative.

The Blind Watchmaker (Dawkins, 1986) is perhaps the best known example of an artificial selection process in computer science. In a design context, human feedback could steer the process without having to give an explanation of why, thus incorporating *tacit* knowledge and drivers. In addition, the motivation for steering the process in a particular direction can change during the process of evolution. As Senatore (2009, p.14) states:

> "Creative evolutionary systems usually do not have a static criteria objective function but rather multiple and dynamic ones to explore alternative search spaces"

The artist William Latham pursued such tacit artificial selection methods in the development of his art. In collaboration with Stephen Todd from IBM in 1980s, the software 'mutator' enabled Latham to *breed* his sculptures and furthermore, explicitly record the process of development (fig.8.2). The process itself became the artwork just as much as the final sculpture itself.

In a design context, Piasecki and Hanna (2011) have explored how meaningful options can be better explored with the aid of evolutionary algorithms with artificial selection. With such a process, the user selects designs from a given generation to move forward to the next, in contrast to an automated process such as roulette wheel selection. Likewise, Steadman(2008) has discussed similar artificial evolutionary processes and the benefit of communicating explicit past design decisions in an evolutionary system. Future work would therefore be the development of an artificial selection engine that could work alongside Embryo within Grasshopper.

8.4. Final Comment

One could argue that as the author of a tool such as Embryo I am imposing a *particular* meta-parametric approach on anyone that wishes to use the component. As a response, I hope that others will begin to to write their own meta-level components that generate topological variants. The tool presented here is not meant to be an end result, but intended to open debate about computational design strategies at the conceptual design stage before the dystopian vision of completely automated building design becomes true.

Embryo is not intended to be a problem solving tool but more like a toy capable of opening up new problems and possibilities to design teams at different levels of abstraction. If it really is turtles all the way down then at least let's try and say hello to them.

Bibliography

Adams, Douglas. 2002. *The salmon of doubt: hitchhiking the galaxy one last time*. Crown.

Adriaenssens, S., Block, P., Veenendaal, D., & Williams, C. (eds). 2014. *Shell Structures for Architecture: Form Finding and Optimization*. London: Taylor & Francis - Routledge.

Aish, Robert, & Woodbury, Robert. 2005. Multi-level interaction in parametric design. *Pages 151–162 of: Smart Graphics*. Springer.

Alexander, Christopher, Ishikawa, Sara, & Silverstein, Murray. 1978. *A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure Series)*. Oxford University Press.

Anderson, Stanford. 1975. *Problem-Solving and Problem-Worrying: Papers delivered at the Architectural Association, London, March 1966*. MIT, Laboratory of Architecture and Planning.

Autodesk. 2007. Parametric Building Modelling: BIM's Foundation (white paper).

Bagger, Anne. 2010. *Plate shell structures of glass*. Ph.D. thesis, PhD Thesis, DTU Civil Engineering, Lyngby (DK) 2010.

Bak, Andreas, Shepherd, Paul, & Richens, Paul. 2012. Intuitive interactive form finding of optimised fabric-cast concrete. *In: Second International Conference on Flexible Formwork (icff2012)*. University of Bath.

Barnes, Michael R. 1999. Form finding and analysis of tension structures by dynamic relaxation. *International journal of space structures*, **14**(2), 89–104.

Bendsoe, Martin Philip. 2003. *Topology optimization: theory, methods and applications*. Springer.

Benjamin, David. 2014. You set up the model. *Colon*, **2**(4), 33–38.

Bentley, Peter, & Kumar, Sanjeev. 1999. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. *Pages 35–43 of: Proceedings of the genetic and evolutionary computation conference*, vol. 1. Morgan Kaufmann.

Bentley, Peter J, & Wakefield, Jonathan P. 1998. Generic evolutionary design. *Pages 289–298 of: Soft Computing in Engineering Design and Manufacturing*. Springer.

Block, Philippe, & Ochsendorf, John. 2007. Thrust Network Analysis: A new methodology for three-dimensional equilibrium. *International Journal of Shell and Spatial Structures*, **155**, 167.

Blockley, David I, & Godfrey, Patrick. 2000. *Doing it differently: systems for rethinking construction*. Inst of Civil Engineers Pub.

Blum, Christian, & Roli, Andrea. 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, **35**(3), 268–308.

Boehm, Barry W. 1988. A spiral model of software development and enhancement. *Computer*, **21**(5), 61–72.

Boers, JWE, & Sprinkhuizen-Kuyper, G. 2001. *Combined Biological Metaphors*. MIT Press, Cambridge, MA. Pages 153–183.

Boeykens, Stefan. 2012. Bridging building information modeling and parametric design. *Page 453 of: EWork and EBusiness in Architecture, Engineering and Construction: Proceedings of the European Conference on Product and Process Modelling 2012, Reykjavik, Iceland, 25-27 July 2012*. CRC Press.

Boulanger, David G. 1961. Program evaluation and review technique. *Advanced Management*, **26**(7), 10.

Bulman, S, Sienz, J, & Hinton, E. 2001. Comparisons between algorithms for structural topology optimization using a series of benchmark studies. *Computers & Structures*, **79**(12), 1203–1218.

Burry, Mark. 2013. *Scripting cultures: Architectural design and programming*. John Wiley & Sons.

Burry, MC. 1996. Parametric design and the Sagrada Familia. *Architecture Research Quarterly, UK*.

Carroll, Sean B. 2005. *Endless forms most beautiful: the new science of evo devo and the making of the animal kingdom*. WW Norton & Company.

Cavieres, Andres, Gentry, Russell, & Al-Haddad, Tristan. 2011. Knowledge-based parametric tools for concrete masonry walls: Conceptual design and preliminary structural analysis. *Automation in Construction*, **20**(6), 716–728.

Chatzikonstantinou, Yannis. 2013. *Hoopsnake for Grasshopper, <url: http://yconst.com/software/hoopsnake/> (accessed 08/13)*.

Checkland, P. 1999. *Systems Thinking, Systems Practice*. John Wiley & Sons, Ltd.

Checkland, Peter. 2000. Soft systems methodology: a thirty year retrospective. *Systems Research and Behavioral Science*, **17**, 11–58.

Chilton, John. 2010. Heinz Isler's Infinite Spectrum: Form-Finding in Design. *Architectural Design*, **80**(4), 64–71.

Chilton, John C. 2000. *Heinz Isler: The Engineer's Contribution to Contemporary Architecture*. Thomas Telford.

Chomsky, Noem. 1957. *Syntactic structures*. Mouton & Co.

Christofides, Nicos. 1975. *Graph theory: An algorithmic approach*. Academic press, New York.

Churchman, C West. 1970. Operations research as a profession. *Management Science*, **17**(2), B–37.

Clune, Jeff, & Lipson, Hod. 2011. Evolving 3d objects with a generative encoding inspired by developmental biology. *ACM SIGEVOlution*, **5**(4), 2–12.

Clune, Jeff, Mouret, Jean-Baptiste, & Lipson, Hod. 2013. The evolutionary origins of modularity. *Proceedings of the Royal Society B: Biological Sciences*, **280**(1755).

Coates, Paul. 2010. *Programming. architecture*. Routledge.

Coates, Paul, Healy, N, Lamb, C, & Voon, W. 1996. The use of Cellular Automata to explore bottom up architectonic rules. *Pages 26–28 of: Eurographics UK Chapter 14th Annual Conference*.

Coates, Paul, Broughton, Terence, & Jackson, Helen. 1999. Exploring three-dimensional design worlds using lindenmayer systems and genetic programming. *Evolutionary design by computers*, 323–341.

Coenders, Jeroen. 2012. NetworkedDesign, Next Generation Infrastructure for Design Modelling. *Pages 39–46 of: Computational Design Modelling*. Springer.

Cohen-Steiner, David, Alliez, Pierre, & Desbrun, Mathieu. 2004. Variational shape approximation. *Pages 905–914 of: ACM Transactions on Graphics (TOG)*, vol. 23. ACM.

Collins, George R. 1963. Antonio Gaudi: Structure and form. *Perspecta*, 63–90.

Cormen, Thomas H, Leiserson, Charles E, Rivest, Ronald L, & Stein, Clifford. 2001. *Introduction to algorithms*. MIT press.

Costa, Luciano da Fontoura Da, & Cesar Jr, Roberto Marcondes. 2000. *Shape analysis and classification: theory and practice*. CRC Press, Inc.

Cramer, Nichael Lynn. 1985. A representation for the adaptive generation of simple sequential programs. *Pages 183–187 of: ICGA*.

Culmann, Karl. 1875. *Die graphische statik*. Vol. 2. Meyer & Zeller (A. Reimann).

Cutler, B., & Whiting, E. 2007. Constrained planar remeshing for architecture. *Pages 11–18 of: Proceedings of Graphics Interface 2007*. ACM.

Darwin, Charles. 1872. *The Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. 6th edn. Nature.

Davis, D. 2013. *Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture*. Ph.D. thesis, School of Architecture and Design & College of Design and Social context, RMIT University.

Davis, D, Pallett, J, & Burry, M. 2011a. The flexibility of logic programming: Parametrically regenerating the Sagrada Família. *Pages 29–38 of: Computer-Aided Architectural Design Research in Asia 2011 (CAADRIA 2011): Circuit Bending, Breaking and Mending*. University of Newcastle.

Davis, Daniel, Burry, Jane, & Burry, Mark. 2011b. Understanding visual scripts: Improving collaboration through modular programming. *International Journal of Architectural Computing*, **9**(4), 361–376.

Davis, Daniel, Burry, Jane, & Burry, Mark. 2011c. Untangling parametric schemata: Enhancing collaboration through modular programming. *In: Proceedings of the 14th international conference on Computer Aided Architectural Design, University of Liege, Liege*.

Dawkins, Richard. 1986. *The blind watchmaker: Why the evidence of evolution reveals a universe without design*. WW Norton & Company.

Dawkins, Richard. 2003. The evolution of evolvability. *On Growth, Form and Computers*, 239–255.

Day, AS. 1965. An introduction to dynamic relaxation. *The engineer*, **219**, 218–221.

De Bono, Edward. 2010. *Lateral thinking: Creativity step by step*. Harper-Collins.

Deb, Kalyanmoy. 2001. Multi-objective optimization. *Multi-objective optimization using evolutionary algorithms*, 13–46.

Deb, Kalyanmoy, Pratap, Amrit, Agarwal, Sameer, & Meyarivan, TAMT. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, **6**(2), 182–197.

DeLanda, Manuel. 2002. Deleuze and the Use of the Genetic Algorithm in Architecture. *Architectural Design*, **71**(7), 9–12.

Derix, Christian. 2008. Genetically Modified Spaces. *Pages 22–26 of:* Littlefield, David (ed), *Space Craft - Developments In Architectural Computing*. RIBA Publishing.

Derix, Christian. 2010. Mediating spatial phenomena through computational heuristics. *Pages 61–66 of: Proceedings of the 30th Annual Conference of the Association for Computer Aided Design in Architecture, ACADIA, New York*.

Derix, Christian, & Izaki, Åsmund. 2013. Spatial Computing for the New Organic. *Architectural Design*, **83**(2), 42–47.

Eastman, Charles M. 1973. Automated space planning. *Artificial intelligence*, **4**(1), 41–64.

Eastman, Chuck. 2009. Automated assessment of early concept designs. *Architectural Design*, **79**(2), 52–57.

Eisenman, Peter, & Somol, Robert E. 1999. *Diagram diaries*. Thames & Hudson.

Eppstein, David, Paterson, Michael S, & Yao, F Frances. 1997. On nearest-neighbor graphs. *Discrete & Computational Geometry*, **17**(3), 263–282.

Evins, R. 2012. *Multi-objective optimisation as an aid to design space exploration for low-carbon buildings*. Ph.D. thesis, University of Bath.

Evins, R., Joyce, S.C., Pointer, P., Sharma, S., Vaidyanathan, R., & Williams, C. 2012. Multi-objective design optimisation : Getting more for less. *Proceedings of the Institution of Civil Engineers - Civil Engineering*, **165**(5), 5–10.

Evins, Ralph. 2013. A review of computational optimisation methods applied to sustainable building design. *Renewable and Sustainable Energy Reviews*, **22**, 230–245.

Faber, Colin. 1963. *Candela, the shell builder*. Reinhold Pub. Corp.

Fischer, Steven R. 2001. *A history of language*. Reaktion books.

Fleischmann, Moritz, & Menges, Achim. 2012. ICD/ITKE Research Pavilion: A case study of multi-disciplinary collaborative computational design. *Computational Design Modelling*, 239–248.

Flöry, Simon, & Pottmann, Helmut. 2010. Ruled surfaces for rationalization and design in architecture. *Pages 103–109 of: Proc. ACADIA*.

Fook, Jan. 2002. Theorizing from practice towards an inclusive approach for social work research. *Qualitative Social Work*, **1**(1), 79–95.

Frazer, John. 1995. *An evolutionary architecture*. Architectural Association Publications.

Fudos, Ioannis. 1995. *Constraint solving for computer aided design*. Ph.D. thesis, Purdue University.

Gardner, Martin. 1970. Mathematical games: The fantastic combinations of John Conway's new solitaire game life. *Scientific American*, **223**(4), 120–123.

Gero, John S. 1990. Design prototypes: a knowledge representation schema for design. *AI magazine*, **11**(4), 26.

Gero, John S, & Kazakov, Vladimir A. 1998. Evolving design genes in space layout planning problems. *Artificial Intelligence in Engineering*, **12**(3), 163–176.

Gill, John, & Johnson, Phil. 2002. *Research methods for managers*. Sage.

Goldberg, David E, & Holland, John H. 1988. Genetic algorithms and machine learning. *Machine learning*, **3**(2), 95–99.

Gould, Stephen Jay. 1977. *Ontogeny and phylogeny*. Harvard University Press.

Grason, John. 1970. *Methods for the Computer-Implemented Solution of a Class of'Floor Plan'Design Problems.* Tech. rept. DTIC Document.

Greenwold, S, Allen, E, & Zalewski, W. 2003. *Active statics*.

Gruber, Petra. 2011. *Biomimetics in architecture: architecture of life and buildings*. Springer.

Gürsel, Dino. 2012. Creative Design Exploration by Parametric Generative Systems in Architecture. *METU JFA*, **1**, 207.

Hanna, Sean. 2007. Automated representation of style by feature space archetypes: distinguishing spatial styles from generative rules. *International Journal of Architectural Computing*, **5**(1), 2–23.

Hanna, Sean, & Turner, Alasdair. 2006. Teaching parametric design in code and construction.

Harding, J., & Lewis, H. 2013 (9). The TRADA Pavilion: A Timber Plate Funicular Shell. *In:* Obrebski, J.B., & Tarczewski, R. (eds), *Proceedings of the International Association for Shell and Spatial Structures (IASS) Symposium 2013*.

Harding, John, & Derix, Christian. 2011. Associative Spatial Networks in Architectural Design: Artificial Cognition of Space using Neural Networks with Spectral Graph Theory. *Pages 305–323 of: Design Computing and Cognition 2010*. Springer.

Harding, John, & Shepherd, Paul. 2011. Structural form finding using zero-length springs with dynamic mass. *In: 2011 IASS Annual Symposium: IABSE-IASS 2011: Taller, Longer, Lighter*. University of Bath.

Harding, John, Joyce, Sam, Shepherd, Paul, & Williams, Chris. 2013. Thinking topologically at early stage parametric design. *Pages 67–76 of: Advances in Architectural Geometry 2012*. Springer.

Heaven, Douglas. 2013. Higher state of mind. *New Scientist*, 32–35.

Henrici, Olaus, & Turner, George Charles. 1903. *Vectors and Rotors: With Applications*. E. Arnold.

Hensel, Michael, Menges, Achim, & Weinstock, Michael. 2004. *Emergence: morphogenetic design strategies*. Wiley-Academy Chichester.

Heumann, Andrew. 2012. *Michael Graves, Digital Visionary: What Digital Design Practice Can Learn From Drawing, <url: http://heumanndesigntech.wordpress.com/2012/09/10/> (accessed 08/13)*.

Heywood, H. 2012. *101 Rules of Thumb for Low Energy Architecture*. RIBA Publishing, London.

Hillier, Bill. 2007. Space is the machine: a configurational theory of architecture.

Hillier, Bill, & Hanson, Julienne. 1984. *The social logic of space*. Vol. 1. Cambridge University Press Cambridge.

Holland, John H. 1992. Genetic algorithms. *Scientific american*, **267**(1), 66–72.

Holzer, Dominik. 2010. Optioneering in Collaborative Design Practice. *International Journal of Architectural Computing*, **8**(2), 165–182.

Holzer, Dominik, Hough, Richard, & Burry, Mark. 2007. Parametric design and structural optimisation for early design exploration. *International Journal of Architectural Computing*, **5**(4), 625–643.

Hooke, Robert. 1675. *A description of helioscopes, and some other instruments*. London.

Hornby, Gregory S. 2003. Generative representations for evolving families of designs. *Pages 1678–1689 of: Genetic and Evolutionary Computation - GECCO 2003, volume 2724 of LNCS*. Springer-Verlag.

Huang, X, & Xie, YM. 2008. A new look at ESO and BESO optimization methods. *Structural and Multidisciplinary Optimization*, **35**(1), 89–92.

Hudson, Roly, Shepherd, Paul, & Hines, David. 2011. Aviva Stadium: A case study in integrated parametric design. *International Journal of Architectural Computing*, **9**(2), 187–204.

Jackson, Michael C. 2000. *Systems approaches to management*. Springer.

Janikow, Cezary Z, & Michalewicz, Zbigniew. 1991. An experimental comparison of binary and floating point representations in genetic algorithms. *Pages 31–36 of: ICGA*.

Jo, Jun H, & Gero, John S. 1998. Space layout planning using an evolutionary approach. *Artificial Intelligence in Engineering*, **12**(3), 149–162.

Johnson-Laird, Philip Nicholas. 1983. *Mental models: Towards a cognitive science of language, inference, and consciousness*. Vol. 6. Harvard University Press.

Jupp, J, & Gero, John S. 2003. Towards computational analysis of style in architectural design. *Pages 1–10 of: IJCAI03 Workshop on Computational Approaches to Style Analysis and Synthesis, IJCAI, Acapulco*. Citeseer.

Kaijima, Sawako, & Michalatos, Panagiotis. 2007. Discretization of continuous surfaces as a design concern. *Pages 901–908 of: Predicting the Future - 25th eCAADe Conference Proceedings*.

Kalay, Yehuda E. 2004. *Architecture's new media: Principles, theories, and methods of computer-aided design*. MIT Press.

Kanellos, Anastasios. 2007. *Topological Self-Organisation: Using a particle-spring system simulation to generate structural space-filling lattices*. M.Phil. thesis, UCL (University College London).

Kay, Alan. 1990. User interface: A personal view. *The art of human-computer interface design*, 191–207.

Kilian, Axel, & Adviser-Nagakura, Takehiko. 2006. Design exploration through bidirectional modeling of constraints.

Kilian, Axel, & Ochsendorf, John. 2005. Particle-spring systems for structural form finding. *Journal of The International Association for Shell and Spatial Structures*, **148**, 77.

Kohonen, Teuvo. 2001. *Self-organizing maps*. Vol. 30. Springer.

Kowaliw, Taras. 2007. *A good number of forms fairly beautiful: an exploration of biologically-inspired automated design*. Ph.D. thesis, Concordia University.

Koza, John R. 1994. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, **4**(2), 87–112.

Kumar, Sanjeev, & Bentley, Peter J. 2000. Implicit evolvability: an investigation into the evolvability of an embryogeny. *Pages 198–204 of:* Whitley, Darrell (ed), *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*.

Kurtz, Cynthia F, & Snowden, David J. 2003. The new dynamics of strategy: Sense-making in a complex and complicated world. *IBM systems journal*, **42**(3), 462–483.

LaCoste, Lucien JB. 1935. A simplification in the conditions for the zero-length-spring seismograph. *Bulletin of the Seismological Society of America*, **25**(2), 176–179.

Lawson, Bryan. 2006. *How designers think: the design process demystified*. Routledge.

Leach, Neil. 2001. *The hieroglyphics of space: Reading and experiencing the modern metropolis*. Routledge.

Leach, Neil. 2009. Digital Morphogenesis. *Architectural Design*, **79**(1), 32–37.

Lewis, Harri. 2011. *Traversing digital matter states to fill irregular volumes*. M.Phil. thesis, University of Bath.

Lewis, Wanda J. 2003. *Tension structures: Form and behaviour*. Thomas Telford.

Liggett, Robin S. 2000. Automated facilities layout: past, present and future. *Automation in Construction*, **9**(2), 197–215.

Liu, Yang, Pottmann, Helmut, Wallner, Johannes, Yang, Yong-Liang, & Wang, Wenping. 2006. Geometric modeling with conical meshes and developable surfaces. *Pages 681–689 of: ACM Transactions on Graphics (TOG)*, vol. 25. ACM.

Mathews, Stanley. 2007. *From agit-prop to free space: the architecture of Cedric Price*. Black Dog Pub. Limited.

Maturana, Humberto R, & Varela, Francisco J. 1980. *Autopoiesis and cognition: The realization of the living*. Vol. 42. Springer.

Mebarki, Abdelkrim, Alliez, Pierre, & Devillers, Olivier. 2005. Farthest point seeding for efficient placement of streamlines. *Pages 479–486 of: Visualization, 2005. VIS 05. IEEE*. IEEE.

Menges, Achim. 2012. Biomimetic design processes in architecture: morphogenetic and evolutionary computational design. *Bioinspiration & biomimetics*, **7**(1).

Menges, Achim, & Ahlquist, Sean. 2011. *Computational design thinking*. Wiley.

Meyer, H von. 1867. Die architektur der spongiosa. *Arch Anat Physiol Wiss Med Reichert DuBois-Reymonds Arch*, **34**, 615–628.

Michalatos, Panagiotis, & Kaijima, Sawako. 2011. Intuitive material distributions. *Architectural Design*, **81**(4), 66–69.

Michell, Anthony George Maldon. 1904. LVIII. The limits of economy of material in frame-structures. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, **8**(47), 589–597.

Miki, Mitsunori, Hashimoto, Masafumi, Fujita, Yoshihisa, Hiroyasu, Tomoyuki, & Shibata, Masaru. 2007. Program search with simulated annealing. *Pages 1754–1754 of: Genetic And Evolutionary Computation Conference: Proceedings of the 9 th annual conference on Genetic and evolutionary computation*, vol. 7.

Miller, J.F., & Harding, S.L. 2008. Cartesian genetic programming. *Pages 2701–2726 of: Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*. ACM.

Miller, Julian. 2001. What bloat? cartesian genetic programming on boolean problems. *Pages 295–302 of: 2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*.

Miller, Julian F, & Thomson, Peter. 2000. Cartesian genetic programming. *Pages 121–132 of: Genetic Programming*. Springer.

Miller, Julian F., & Thomson, Peter. 2003. A developmental method for growing graphs and circuits. *Pages 93–104 of: Proceedings of the 5th international conference on Evolvable systems: from biology to hardware*. ICES'03. Berlin, Heidelberg: Springer-Verlag.

Mitchell, Melanie. 1998. *An introduction to genetic algorithms*. MIT press.

Mitchell, William J. 1990. *The logic of architecture: Design, computation, and cognition*. MIT press.

Moussavi, Farshid. 2011. Parametric software is no substitute for parametric thinking. *The Architectural Review*.

Mueller, Volker. 2011. Distributed Perspectives for Intelligent Conceptual Design. *Distributed Intelligence in Design*, 16–26.

Negroponte, Nicholas. 1969. Toward a Theory of Architecture Machines. *Journal of Architectural Education*, 9–12.

Nervi, Pier Luigi. 1965. *Aesthetics and technology in building*. Harvard Univ Pr.

Norman, Donald A. 1990. *Cognitive artifacts*. Department of Cognitive Science, University of California, San Diego.

Otto, Frei, & Rasch, Bodo. 1995. *Finding form: towards an architecture of the minimal*.

Pahl, Gerhard, Wallace, Ken, & Blessing, Lucièenne. 2007. *Engineering design: a systematic approach*. Vol. 157. Springer.

Paulson, Boyd C. 1976. Designing to reduce construction costs. *Journal of the Construction Division*, **102**(4), 587–592.

Perec, Georges. 1997. *Species of spaces and other pieces: Georges Perec*. Penguin.

Peters, Brady. 2007. The Smithsonian Courtyard Enclosure: a case-study of digital design processes.

Phillips, Roland. 2008. *Plan of Work: Multi-disciplinary Services*. RIBA Enterprises.

Piasecki, Michal, & Hanna, Sean. 2011. A Redefinition of the Paradox of Choice. *Pages 347–366 of: Design Computing and Cognition 2010*. Springer.

Plume, Jim, & Mitchell, John. 2007. Collaborative design using a shared IFC building model - Learning from experience. *Automation in Construction*, **16**(1), 28–36.

Poincaré, Henri. 1914. *Science and method*. DoverPublications. com.

Poli, Riccardo, Langdon, W William B, McPhee, Nicholas F, & Koza, John R. 2008. *A field guide to genetic programming*. Lulu. com.

Preisinger, Clemens. 2013. Linking structure and parametric geometry. *Architectural Design*, **83**(2), 110–113.

Prusinkiewicz, Przemyslaw, & Lindenmayer, Aristid. 1990. *The algorithmic beauty of plants*. Springer-Verlag.

Puusepp, Renee. 2011. *Generating circulation diagrams for architecture and urban design using multi-agent systems*. Ph.D. thesis, University of East London.

Rafiq, Y., & Beck, M. 2008. A decision support tool for multidisciplinary conceptual design. *The Structural Engineer*, 37–42.

Rechenberg, Ingo. 1973. Evolution Strategy: Optimization of Technical systems by means of biological evolution. *Fromman-Holzboog, Stuttgart*, **104**.

Ren, Z, Yang, F, Bouchlaghem, NM, & Anumba, CJ. 2011. Multidisciplinary collaborative building design - A comparative study between multi-agent systems and multi-disciplinary optimisation approaches. *Automation in Construction*, **20**(5), 537–549.

Richens, Paul. 2011. *Distributed intelligence in design*. Wiley Online Library. Chap. Foreword, pages x–xi.

Rittel, Horst WJ, & Webber, Melvin M. 1973. Dilemmas in a general theory of planning. *Policy sciences*, **4**(2), 155–169.

Roberts, Stephen G, & Turega, Mike. 1995. Evolving neural network structures: An evaluation of encoding techniques. *Pages 96–99 of: Artificial Neural Nets and Genetic Algorithms*. Springer.

Robson, Colin. 2002. *Real world research: A resource for social scientists and practitioner-researchers*. Vol. 2. Blackwell Oxford.

Roudsari, Mostapha Sadeghipour, Pak, Michelle, & Smith, Adrian. 2013. ladybug: A Parametric Environmental Plugin For Grasshopper To Help Designers Create An Environmentally-Conscious Design. *Pages 26–28 of: proceedings of bs2013: 13th conference of international building performance association, chambery, france, august*.

Rowe, Peter G. 1991. *Design thinking*. The MIT Press.

Russell, Stuart Jonathan, Norvig, Peter, Canny, John F, Malik, Jitendra M, & Edwards, Douglas D. 1995. *Artificial intelligence: a modern approach*. Vol. 74. Prentice hall Englewood Cliffs.

Rutten, David. 2013. Galapagos: On the Logic and Limitations of Generic Solvers. *Architectural Design*, **83**(2), 132–135.

Schön, Donald A. 1983. *The reflective practitioner: How professionals think in action*. Vol. 5126. Basic books.

Schrödinger, Erwin. 1992. *What is life?: With mind and matter and autobiographical sketches*. Cambridge University Press.

Schumacher, Patrik. 2009. Parametricism: A new global style for architecture and urban design. *Architectural Design*, **79**(4), 14–23.

Schwartz, Barry, & Ward, Andrew. 2004. Doing better but feeling worse: The paradox of choice. *Positive psychology in practice*, 86–104.

Senatore, Gennaro. 2009. *Morphogenesis of Spatial Configurations*. M.Phil. thesis, Universiy of East London.

Shea, Kristina, Aish, Robert, & Gourtovaia, Marina. 2005. Towards integrated performance-driven generative design tools. *Automation in Construction*, **14**(2), 253–264.

Shepherd, Paul. 2009. Digital architectonics in practice. *In: 27th eCAADe Conference*. University of Bath.

Shepherd, Paul, Hudson, Roly, & Hines, David. 2011. Aviva Stadium: A parametric success. *International Journal of Architectural Computing*, **9**(2), 167–186.

Simmonds, Tristan, Self, Martin, & Bosia, Daniel. 2006. Woven surface and form. *Architectural Design*, **76**(6), 82–89.

Simon, Herbert A. 1962. The architecture of complexity. *Proceedings of the American philosophical society*, **106**(6), 467–482.

Simpson, Timothy W, Rosen, David, Allen, Janet K, & Mistree, Farrokh. 1998. Metrics for assessing design freedom and information certainty in the early stages of design. *Transactions - American society of mechanical engineers journal of mechanical design*, **120**, 628–635.

Sims, K. 1994. Evolving 3D morphology and behavior by competition. *Artificial life*, **1**(4), 353–372.

Smith, Susan. 2008. *Engineering sidra trees. <url:http://www.architectureweek.com/2008/0227/> (accessed 08/13)*.

Stanley, Kenneth O, & Miikkulainen, Risto. 2003. A taxonomy for artificial embryogeny. *Artificial Life*, **9**(2), 93–130.

Steadman, Philip. 1983. *Architectural morphology: an introduction to the geometry of building plans*. Pion London.

Steadman, Philip. 2008. *The Evolution of Designs: Biological analogy in architecture and the applied arts*. Routledge.

Stiny, George, Mitchell, William J, *et al.* 1978. The palladian grammar. *Environment and Planning B*, **5**(1), 5–18.

Succar, Bilal. 2009. Building information modelling framework: A research and delivery foundation for industry stakeholders. *Automation in Construction*, **18**(3), 357–375.

Swinson, Peter SG. 1982. Logic programming: A computing tool for the architect of the future. *Computer-aided design*, **14**(2), 97–104.

Thaler, Richard H, & Sunstein, Cass R. 2008. *Nudge: Improving decisions about health, wealth, and happiness*. Yale University Press.

Todd, Stephen, & Latham, William. 1994. Evolutionary art and computers.

Turing, AM. 1952. The Chemical Basis of Morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, **237**(641), 37–72.

Turk, Greg. 1992. Re-tiling polygonal surfaces. *Pages 55–64 of: ACM SIGGRAPH Computer Graphics*, vol. 26. ACM.

Turrin, Michela, von Buelow, Peter, Stouffs, Rudi, & Kilian, Axel. 2010. Performance-oriented design of large passive solar roofs. *A method for the integration of parametric modeling and genetic algorithms. in: Proceedings of eCAADe*, 321–330.

Turrin, Michela, von Buelow, Peter, & Stouffs, Rudi. 2011. Design explorations of performance driven geometry in architectural design using parametric modeling and genetic algorithms. *Advanced Engineering Informatics*, **25**(4), 656–675.

van Leeuwen, Jan. 1991. *Graph algorithms, Handbook of theoretical computer science (vol. A): algorithms and complexity*.

Vassilev, Vesselin K, & Miller, Julian F. 2000. The advantages of landscape neutrality in digital circuit evolution. *Pages 252–263 of: Evolvable systems: from biology to hardware*. Springer.

Venturi, Robert, Brown, Denise Scott, & Izenour, Steven. 1972. *Learning from Las Vegas*. Vol. 102. MIT press Cambridge, MA.

Verbitsky, Oleg. 2008. On the obfuscation complexity of planar graphs. *Theoretical Computer Science*, **396**(1), 294–300.

Wagner, Gunter P, & Altenberg, Lee. 1996. Perspective: Complex adaptations and the evolution of evolvability. *Evolution*, 967–976.

Wakefield, DS. 1985. Tensyl: an integrated CAD approach to stressed membrane structures. *In: Proc. Second Inter. Conf. on Civil and Structural Eng., Edinburgh*.

Wang, Wenping, Liu, Yang, Yan, Dongming, Chan, Bin, Ling, Ruotian, & Sun, Feng. 2008. Hexagonal meshes with planar faces. *Dept. of CS, HKU, Tech. Rep*.

Watts, Duncan J, & Strogatz, Steven H. 1998. Collective dynamics of small-world networks. *Nature*, **393**(6684), 440–442.

Weinstock, Michael. 2010. *The architecture of emergence: the evolution of form in nature and civilisation*. Wiley London/Chichester.

Werner, James Cunha. 2001. The Physics behind Genetic Programming.

Wester, T. 1992. An Approach to a Form and Force Language Based on Structural Dualism. *Pages 14–24 of: Proceedings of the 1st International Colloquium on Structural Morphology, IASS Working Group*.

Wester, Ture. 2002. Nature teaching structures. *International Journal of Space Structures*, **17**(2), 135–147.

Williams, C. 1986. Defining and designing curved flexible tensile surface structures. *The mathematics of surfaces*, 143–177.

Williams, Chris JK, & Hudson, Roly. 2011. Distributed intelligence or a simple coherent mental model? *Distributed Intelligence In Design*, 27.

Williams, Christopher JK. 2001. The analytic and numerical definition of the geometry of the British Museum Great Court Roof.

Winslow, P. 2009. *Synthesis and optimisation of free-form grid structures*. Ph.D. thesis, Thesis (PhD). University of Cambridge.

Winslow, Peter, Pellegrino, Sergio, & Sharma, Shrikant B. 2007. Mapping two-way grids onto free-form surfaces. *Pages 3–6 of: International association of shell and spatial structures symposium, Venice*.

Wolfram, Stephen. 1984. Universality and complexity in cellular automata. *Physica D: Nonlinear Phenomena*, **10**(1), 1–35.

Wolpert, Lewis. 1991. *The triumph of the embryo*. Oxford University Press.

Woodbury, Robert. 2010. Elements of parametric design.

Woodbury, Robert, Aish, Robert, & Kilian, Axel. 2007. Some patterns for parametric modeling. *Pages 222–229 of: 27th Annual Conference of the Association for Computer Aided Design in Architecture*.

Woodward, John R. 2006. Complexity and cartesian genetic programming. *Pages 260–269 of: Genetic Programming*. Springer.

Xie, Y Mike, & Steven, Grant P. 1997. *Basic evolutionary structural optimization*. Springer.

Zimmer, Henrik, Campen, Marcel, Herkrath, Ralf, & Kobbelt, Leif. 2013. Variational tangent plane intersection for planar polygonal meshing. *Pages 319–332 of: Advances in Architectural Geometry 2012*. Springer.

Zitzler, Eckart, Laumanns, Marco, Thiele, Lothar, Zitzler, Eckart, Zitzler, Eckart, Thiele, Lothar, & Thiele, Lothar. 2001. *SPEA2: Improving the strength Pareto evolutionary algorithm*.

Relevant Publications by Author

Internal publications (at Ramboll):

Harding, J., 2010. Principal Stress The development of an Exhibition Piece for the Ramboll UK Office. *In: 2010 Ramboll Technical Forum Proceedings*, Ramboll.

Harding, J., 2012. Finding Funicular Form with Weak Springs. *In: Ramboll Research and Development Forum Proceedings*, Ramboll.

External publications:

Harding, J. and Derix, C., 2011. Associative spatial networks in architectural design: Artificial cognition of space using neural networks with spectral graph theory. *In: Design Computing and Cognition '10*. New York: Springer Science and Business Media, pp. 305-323.

Harding, J. and Shepherd, P., 2011. Structural form finding using zero-length springs with dynamic mass. *In: 2011 IASS Annual Symposium: IABSE-IASS 2011: Taller, Longer, Lighter, 20-23 September 2011, London*.

Harding, J., Joyce, S., Shepherd, P. and Williams, C., 2012. Thinking Topologically at Early Stage Parametric Design. *In: Proceedings of Advances in Architectural Geometry 2012, Paris*.

Harding, J. & Lewis, H. 2013. The TRADA Pavilion – A Timber Plate Funicular Shell. *In: Proceedings of the 2013 IASS Annual Symposium: Beyond the Limit of Man.*

Joyce, S. & Harding, J., 2013. Generator 2.0. *In: Proceedings of the 4th Design Modelling Symposium, Berlin.*

Melville, S.; Lewis, H. & Harding, J. 2013. TRADA Pavilion - Searching for Innovation and Elegance in Complex Forms Supported by Physical and Software Prototyping. *In: Stacey, M. (Ed.) Prototyping Architecture: The Conference Papers, Building Centre Trust, London, 2013*, pp. 277-289.

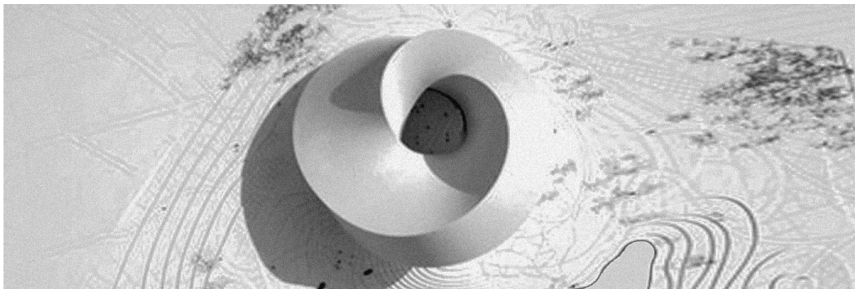
Part VI.

Appendix

A. Case Study Details

The case study projects are ordered chronologically and this appendix contains the following information:

- Project information.
- Internal collaborators.
- External collaborators.
- Software used on project.
- Short description of work conducted.



[01Anl]

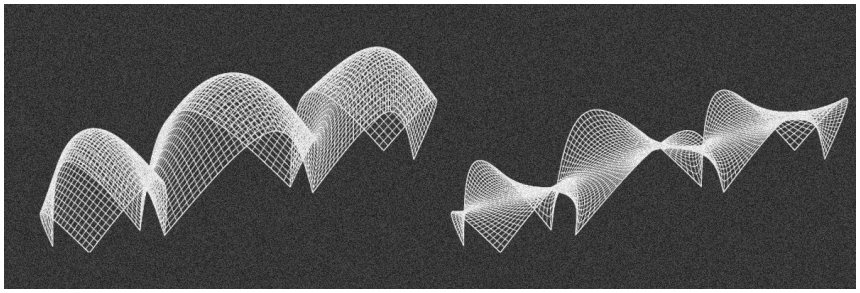
Astana National Library, 2009

Project Information

- Internal Collaborators: Structures: Stephen Melville, Duncan Horswill, Tom Foley. Façades: Neesha Gopal, Will Stevens, Mark Pniewski.
- External Collaborators: Bjarke Ingels Group Architects
- Location: Astana, Kazakhstan
- Floor area: 45,000m^2
- Project type: Public building
- Software/language used: Rhino Grasshopper

Description of work

Ramboll were employed by the architect to provide structural engineering and facade services. The initial geometric concept of the Möbius strip had already been developed by the architect, inspired by some of their own recent work using a continuous circulation concept. This was sent via a parametric model definition. The form was geometrically pure and frozen which enabled stakeholders to easily use a common model in order to conduct experiments, such as computational heuristics that embedded structural and fabrication logic in order to improve on the current design. In the end, even though improvements were made, the facade cost was still high due to the underlying geometry.



[02Lbr]

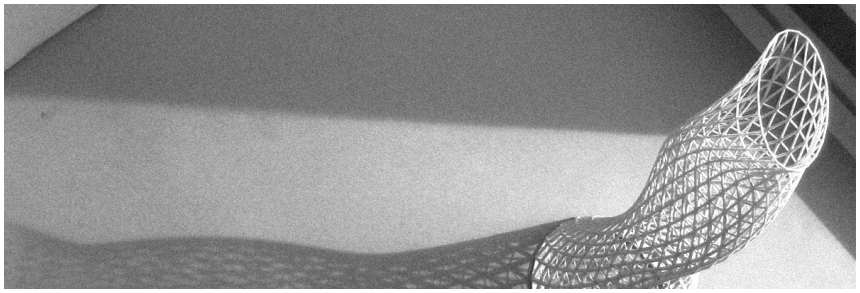
Lusail Bridge Roof, 2009

Project Information

- Internal Collaborators: Bridges: Stephen James.
- External Collaborators: None
- Location: Lusail, Qatar.
- Bridge span: 120m
- Project type: Footbridge
- Software/language used: Microstation Visual Basic

Description of work

Working with our in-house bridges team, I provided the form-finding of the doubly curved roof form using the Laplacian smoothing process. The geometry was imported as a surface, and due to the algorithm being written directly in a CAD environment (Microstation), could be be easily exported to the architect in the same format.



[03Cht]

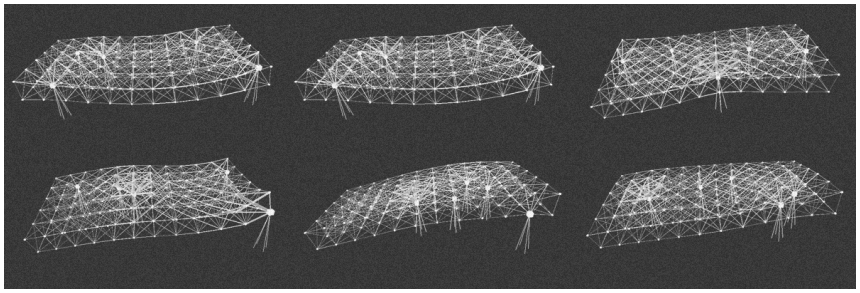
Cheongna Tower, 2009

Project Information

- Internal Collaborators: Structures: Stephen Melville.
- External Collaborators: Various Architects (Jim Dodson)
- Location: Cheongna, Korea
- Tower height: 450m
- Project type: Observation tower (Competition)
- Software/language used: Rhino Grasshopper, Generative Components

Description of work

This project began life as a hyperboloid tower concept, which then developed into a form made from lofted ellipses that rotated. It is an example where I used the concept of a parametric model with multi-objective optimisation to find a collection of *good* designs. In the end, these 'optimial' designs were actually quite poor because of the underlining geometry of the tower. Although aesthetically pleasing, the form created structural inefficiencies that could not be resolved.



[04Bhc]

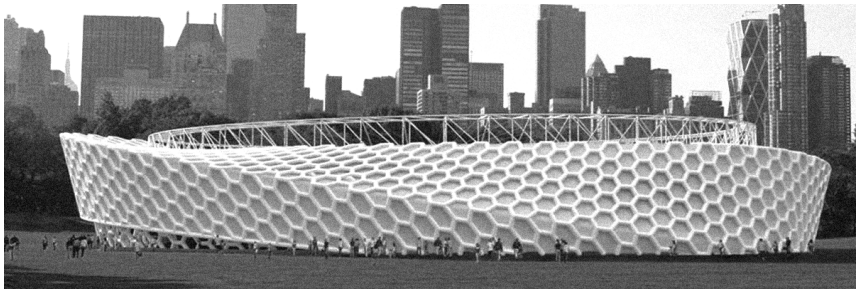
Bath House Columns, 2010

Project Information

- Internal Collaborators: Structures: Iain Sproat, Kevin Hares.
- External Collaborators: None (directly)
- Location: London, UK.
- Floor area: 20,000m^2
- Project type: Office/Commercial
- Software/language used: Java/Processing

Description of work

This project used a simple brute-force method for solving a practical engineering problem. In this case, it was to locate a given number of supports in order to minimise the deflection of a roof space frame. Structural analysis was conducted at each iteration using the dynamic relaxation method implemented in a Java application, with the best design recorded if an improvement was made on the current best.



[05Aat]

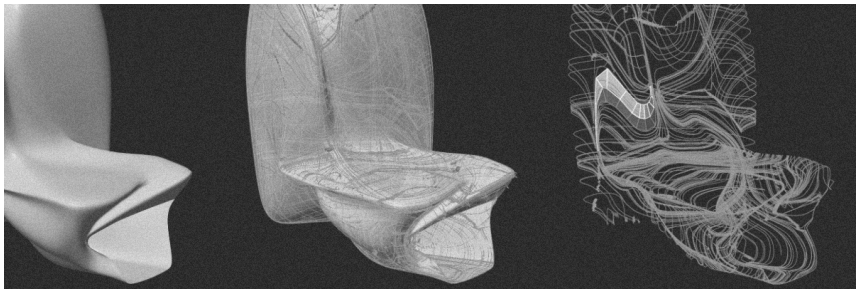
Arts Alliance Travelling Theatre, 2010

Project Information

- Internal Collaborators: Structures: Stephen Melville.
- External Collaborators: Various Architects (Jim Dodson)
- Location: (mobile venue)
- Venue size: 450m^2
- Project type: Performance venue
- Software/language used: Generative Components.

Description of work

The Arts Alliance Project is a case study in the traditional use of Parametric Design. We investigated different surface patterns on a given doubly curved form using the parametric model, before arriving on a good compromise with the architect between visual appearance and fabrication cost. The final design was then exported to CAD for further detailed work.



[06Nhq]

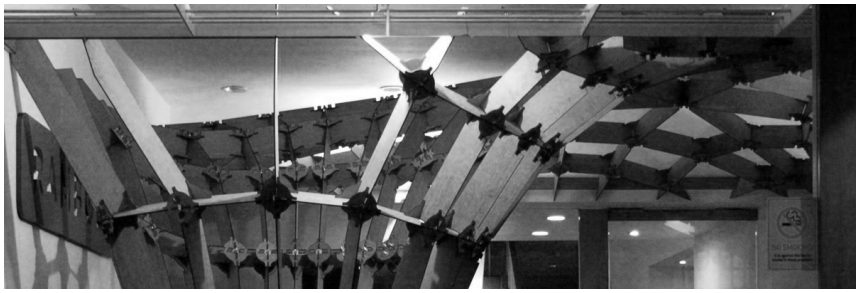
National Holdings Headquarters, 2010

Project Information

- Internal Collaborators: Structures: Stephen Melville, Duncan Horswill, Harri Lewis. Façades: Neesha Gopal, Mark Pniewski.
- External Collaborators: Zaha Hadid Architects
- Location: Abu Dhabi, UAE
- Floor area: 44,000m^2
- Tower height: 14 Storey
- Project type: Office/Commercial
- Software/language used: Rhino Grasshopper, C#

Description of work

The NHHQ was a complex doubly curved building envelope which required post-rationalisation to reduce facade cost. Hot bent doubly curved glass is expensive to manufacture, and hence a method to produce flat quadrilaterals was adopted. This involved using the facade surface principal curvature field to generate a Planar Quadrilateral (PQ) panelling layout.



[07Foy]

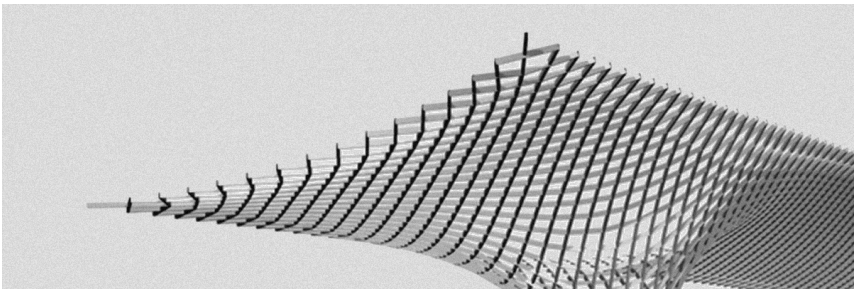
London Foyer Sculpture, 2010-11

Project Information

- Internal Collaborators: Structures: Duncan Horswill, Stephen Melville, James Solly.
- External Collaborators: None
- Location: Ramboll UK Head Office, London, UK
- Size: 5m x 5m x 3m
- Project type: Timber gridshell sculpture
- Software/language used: Generative Components, Java/Processing

Description of work

The first independent project by Ramboll Computational Design. See Appendix D for a detailed explanation.



[08Urb]

Urban Bubble Gridshell, 2010

Project Information

- Internal Collaborators: Structures: Yanchee Lau, Stephen Melville.
- External Collaborators: None
- Location: London.
- Size: Unknown
- Project type: Continuous timber gridshell
- Software/language used: Microstation Visual Basic

Description of work

The application of Laplacian smoothing to generate a doubly curved timber soffit around four external columns.

[09Nuk]

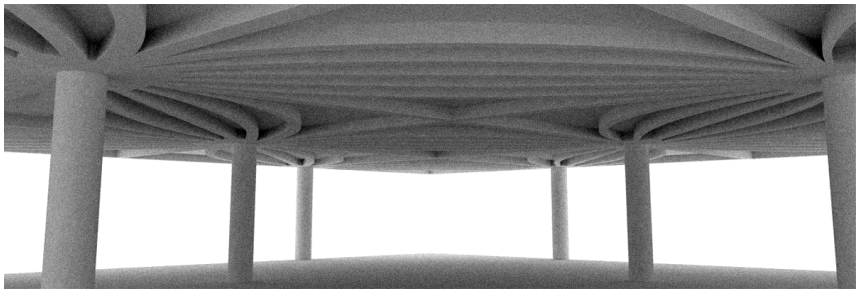
National Gallery of Greenland, 2010

Project Information

- Internal Collaborators: Structures: Duncan Horswill, Ross Smith, Tom Foley.
- External Collaborators: Bjarke Ingels Group Architects
- Location: Nuuk, Greenland
- Size: 3000m^2
- Project type: Public building
- Software/language used: Rhino Grasshopper

Description of work

With our structural engineers in Bristol, a ribbed soffit was generated that followed the lines of principal stress in order to improve the efficiency of the roof structure.



[10Vik]

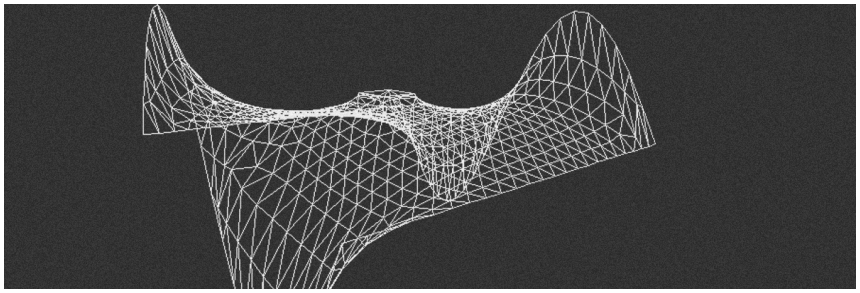
Viikki Synergy Building, 2010

Project Information

- Internal Collaborators: Structures: Stephen Melville, Duncan Horswill.
- External Collaborators: (Ramboll Denmark)
- Location: Viikki, Helsinki, Finland
- Size: 13,500m^2
- Project type: Office building
- Software/language used: Rhino Grasshopper

Description of work

With our structural engineers in Denmark, a ribbed soffit was generated that followed the lines of principal stress in order to improve the efficiency of the roof structure. In this case the repeatable units and regular column grid gave rise to similar results to Pier Luigi Nervi's famous Gatti Wool Factory (1951).



[11Gfp]

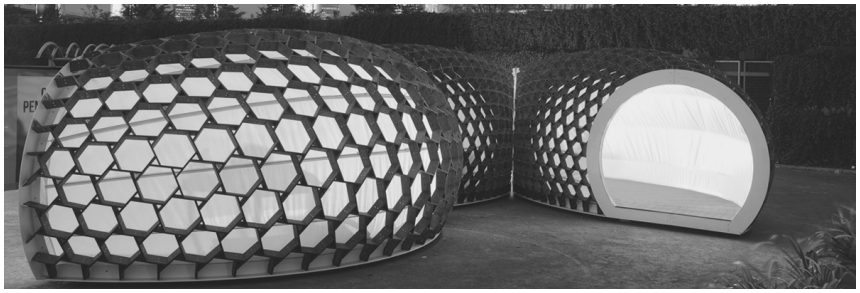
Garden Festival Pavilion, 2011

Project Information

- Internal Collaborators: Structures: Stephen Melville.
- External Collaborators: None
- Location: Unknown
- Size: 170m^{2}fabric structure.
- Project type: Small temporary shelter
- Software/language used: Java/Processing

Description of work

The form-finding of a minimal surface using dynamic relaxation for a continuum utilising the triple-force method.



[12Kre]

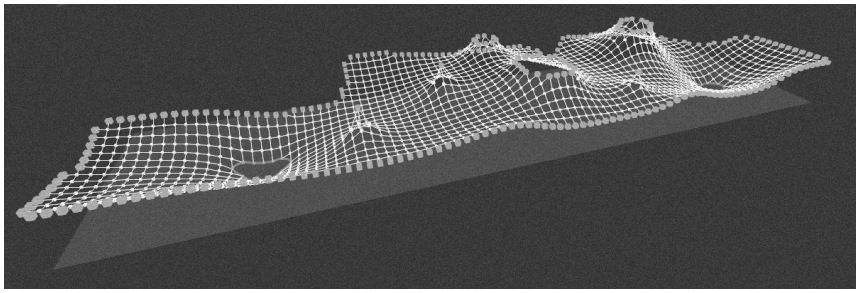
KREOD Pavilion, 2011-12

Project Information

- Internal Collaborators: Structures: Harri Lewis, Stephen Melville.
- External Collaborators: Pavilion Architecture
- Location: Greenwich Peninsula, London, UK
- Project type: Temporary pavilion (3 identical structures)
- Software/language used: Rhino Grasshopper, Microstation VBA
- Involvement: RIBA Stage D

Description of work

The architect's initial design created complex joints at each node due to the surface form and the diagrid discretisation method. We attempted the principal curvature approach for generating planar quadrilaterals and clean node connections by developing software and handing it to the architect. This method was not adopted due in part because I was attempting to apply a heuristic to constraint the designs for my own benefit, which overruled the architect. Although technologically sound, the approach was unsuccessful because of the way it was integrated in the design process. The final design itself was a success, showing that trying to force a particular heuristic (in software) early on was probably the wrong approach in a collaborative environment.



[13Rig]

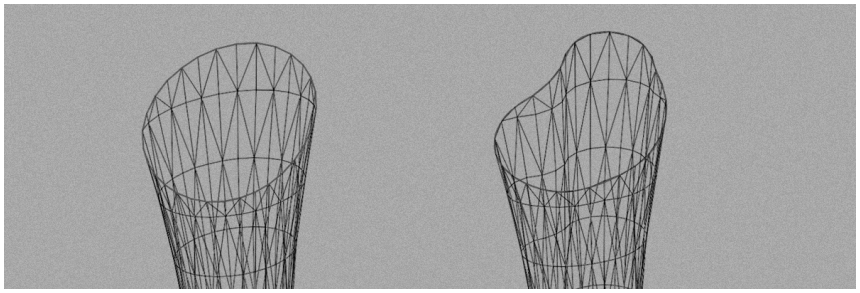
Air Baltic Terminal, Riga International Airport, 2011

Project Information

- Internal Collaborators: Structures: James Norman.
- External Collaborators: Allianss Arhitektid, Estonia.
- Location: Riga, Latvia
- Project type: Airport terminal
- Footprint: 25,000m^2
- Software/language used: Java/Processing

Description of work

Similar to the KREOD pavilion [12Kre], an attempt to develop a software application and hand it to the architect to constrain the design. The project was successful as the imposed heuristic meant the design was similar to the original intentions of the architect.



[14Mvt]

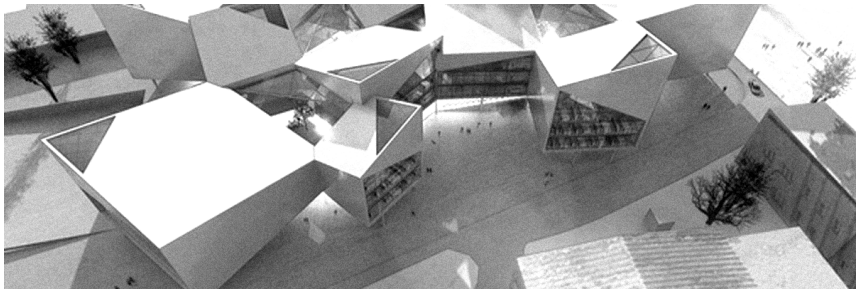
Markham Vale Tower, 2011

Project Information

- Internal Collaborators: Structures: Tom Foley.
- External Collaborators: Local artist
- Location: Markham Vale, Chesterfield, UK
- Project type: Sculpture
- Tower height: 40m
- Software/language used: Java/Processing

Description of work

An artist sculpture with a relaxation method applied to help reduce bending moment in the structure. The resulting geometry was similar in shape to the original whilst improving the structure.



[15Tth]

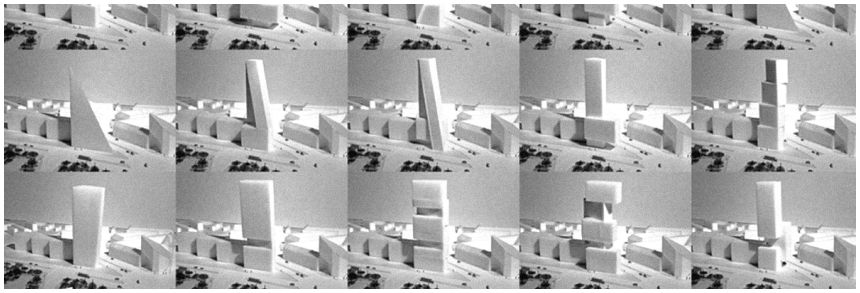
Tallinn Town Hall, 2011

Project Information

- Internal Collaborators: Structures: Harri Lewis, Stephen Melville, Tom Foley, Duncan Horswill.
- External Collaborators: Bjarke Ingels Group Architects
- Location: Tallinn, Estonia
- Project type: Administrative building
- Floor area: 28,000m^2
- Software/language used: Topopt

Description of work

The project contained a series of cantilevered boxes supported by large steel trusses within their facades. The final layout of these trusses was assisted by using a topology optimisation solver; not to find the perfect result necessarily, but rather to guide the design team towards a *good* design whilst balancing other objectives such as fabrication complexity.



[16Esc]

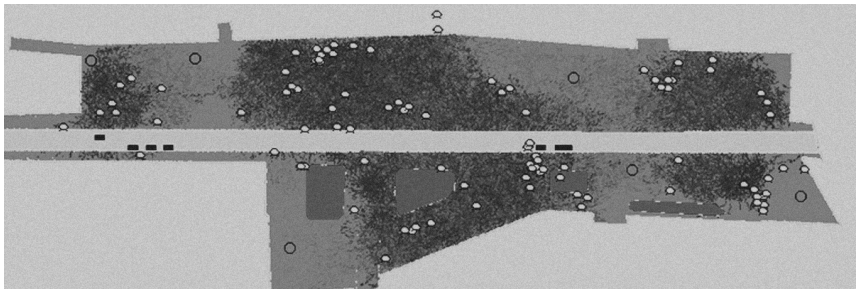
Escher Tower, 2011-12

Project Information

- Internal Collaborators: Structures: Duncan Horswill. Building Physics: Tom Shilton.
- External Collaborators: Bjarke Ingels Group Architects
- Location: Copenhagen, Denmark
- Project type: Hotel
- Floor area: 15,000m^2
- Tower height: 100m
- Software/language used: Rhino Grasshopper, Ecotect.

Description of work

A first attempt at using parametric design at the early stage of design in collaboration with an architect. Our involvement was not a success because although we had the analysis tools, existing top-down modelling processes could not quickly jump between different building topologies. This project led to a rejection of parametric design in favour of a procedural modelling approach.



[17Fsc]

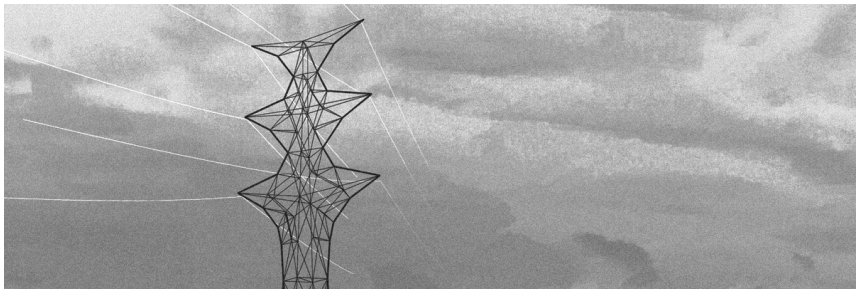
Forgotten Spaces Competition, 2011

Project Information

- Internal Collaborators: Structures: Stephen Melville.
- External Collaborators: Texere Studio Architects
- Location: Kentish Town, London, UK
- Project type: Urban regeneration
- Software/language used: Java/Processing

Description of work

The use of an agent based model to simulate the effect of flocking sheep when introduced to an urban environment. The positioning of feeding stations throughout the site was influenced by feedback from the simulation.



[18Pyl]

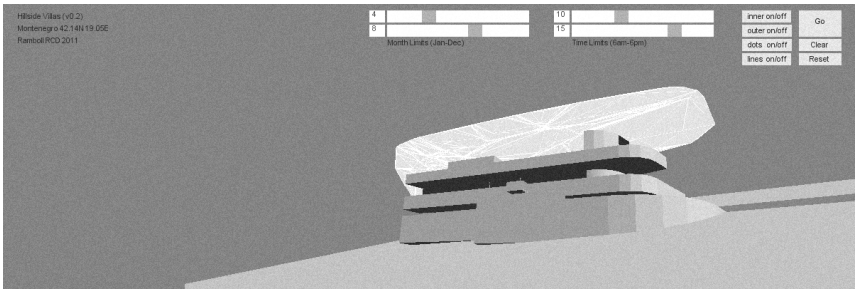
RIBA Pylon Competition, 2011

Project Information

- Internal Collaborators: Structures: Stephen Melville, Duncan Horswill.
- External Collaborators: None
- Location: UK
- Project type: Electricity pylon ideas competition
- Software/language used: Java/processing

Description of work

Structural optimisation of a new pylon structure using a genetic algorithm.



[19Mjb]

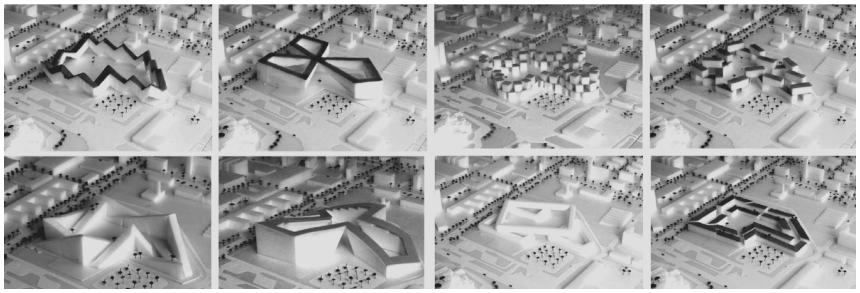
Maljevik Bay Resort, 2011

Project Information

- Internal Collaborators: Structures: Stephen Melville, Duncan Horswill. Building Physics: Tom Shilton.
- External Collaborators: Foster + Partners
- Location: Maljevik Bay, Montenegro
- Project type: Resort Villas and Apartments.
- Software/language used: Java/processing

Description of work

Working with passive solar design principles in the design of several Villas all orientated slightly differently on a site. A computational method was developed that accumulated the incident sun angles for summer months and used this to form an optimal roof design. Each roof allowed a large amount of sun light for winter months, but restricted the sun in the summer months to avoid solar gain.



[20Eni]

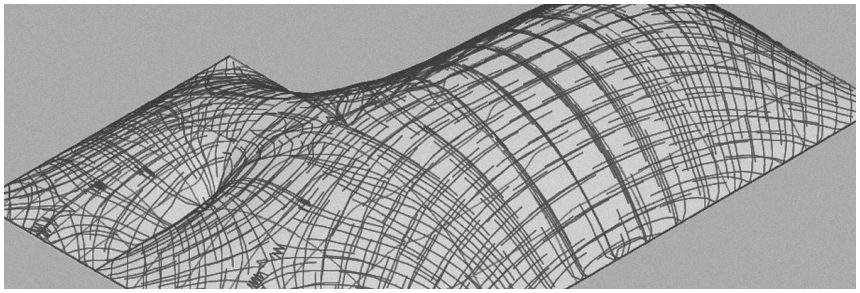
ENI Exploration and Production HQ, 2011

Project Information

- Internal Collaborators: Structural engineering: Duncan Horswill, Building Physics: Tom Shilton.
- External Collaborators: Bjarke Ingels Group Architects
- Location: Milan, Italy
- Floor area: 20,000m^2
- Project type: Office/Commercial (Competition)
- Software/language used: Java/Processing

Description of work

Competition project with BIG, described fully in Chapter 5. I developed a computational approach that generated forms and integrated real-time analysis. The black-box complexity of the geometry generation meant meaningful interaction was hard and the architect did not eventually adopt the approach. The issue of control was an important factor in our collaboration and led to a return to parametric modelling in my research.



[21Chs]

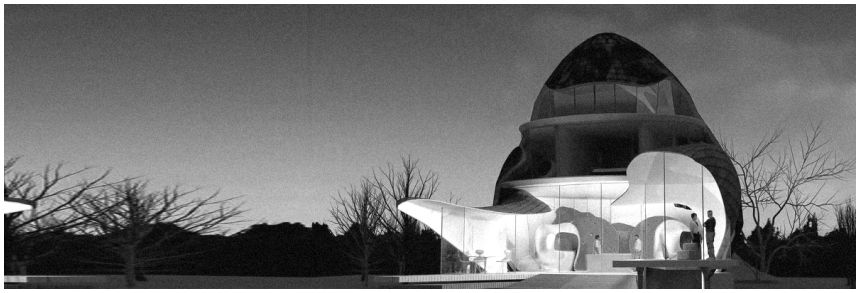
Cockaigne House Shell, 2011

Project Information

- Internal Collaborators: Structures: Rob Harrold.
- External Collaborators: None
- Location: Hatfield, UK
- Footprint: 50m^2
- Project type: Timber gridshell structure
- Software/language used: Java/processing, Rhino Grasshopper

Description of work

Form-found timber roof for a private client with a dominant structural heuristic. As I was the only designer involved on the project, a clear computational approach was easy to achieve.



[22Orc]

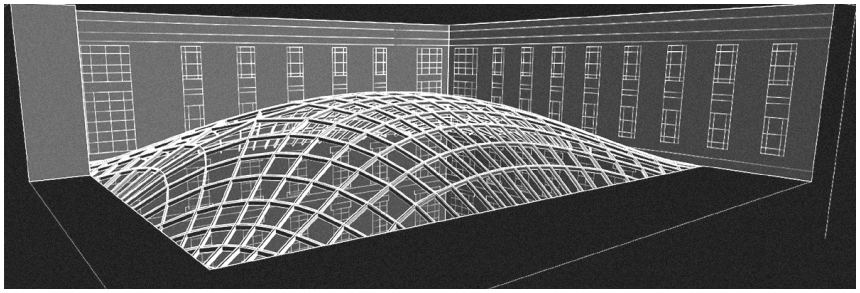
Orchid House, 2011

Project Information

- Internal Collaborators: Structures: Duncan Horswill, Andreas Bak.
- External Collaborators: Featherstone Young Architects, Cowley Timber.
- Location: Cirencester, UK
- Project type: Private house
- Software/language used: Microstation Visual Basic

Description of work

A house which had already been designed that we had to 'make work' structurally. For the facade, the use of a principal curvature approach discretised the architect's fixed surface inefficiently. Small modifications to the underlying surface however gave rise to drastic changes in the discretisation pattern for which the architect was happy to compromise. This inspired wrapping a similar heuristic and handing it to the architect on the KREOD pavilion A.



[23Shr]

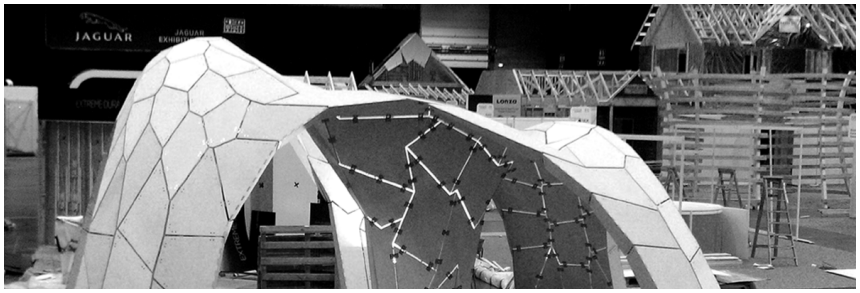
Senate House Roof, 2012

Project Information

- Internal Collaborators: Structures: Kate Waldron, Duncan Horswill. Façade engineering: Neesha Gopal.
- External Collaborators: Carey Jones Architects
- Location: London, UK
- Size: 28m x 21m (perimeter)
- Project type: Glass atrium roof
- Software/language used: Processing/Java, Rhino Grasshopper

Description of work

Form-finding of a steel gridshell and discretisation to planar quadrilateral panels.



[24Tra]

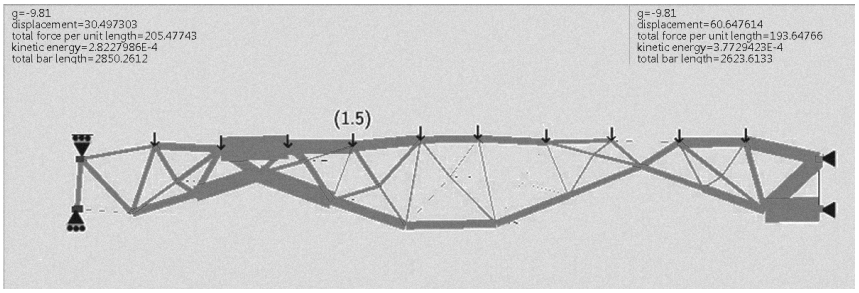
TRADA Pavilion, 2012

Project Information

- Internal Collaborators: Structures: Stephen Melville, Harri Lewis, Duncan Horswill, Andreas Bak, Emily Scoones.
- External Collaborators: Timber Research and Development Association (TRADA)
- Location: Various
- Project type: Timber Plate Shell
- Size: 8m x 6m x 4m
- Software/language used: Rhino Grasshopper, Java/Processing, C#

Description of work

Form-found trade fair pavilion made from timber panels for the Timber Research and Development Association (TRADA). See Appendix E for more detailed description.



[25Ves]

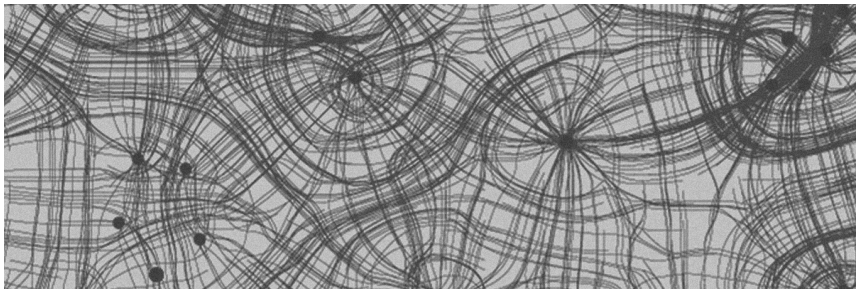
Vestas Blade Technology Centre, 2012

Project Information

- Internal Collaborators: Structures: Duncan Horswill.
- External Collaborators: None
- Location: Isle of Wight, UK
- Project type: Offices/Manufacturing
- Floor area: 29,000m^2
- Software/language used: Java/Processing, C#

Description of work

2D topology optimisation for a repeated truss form. I wrote a bespoke Java application to optimise different topological structures using a genetic algorithm.



[26Hcl]

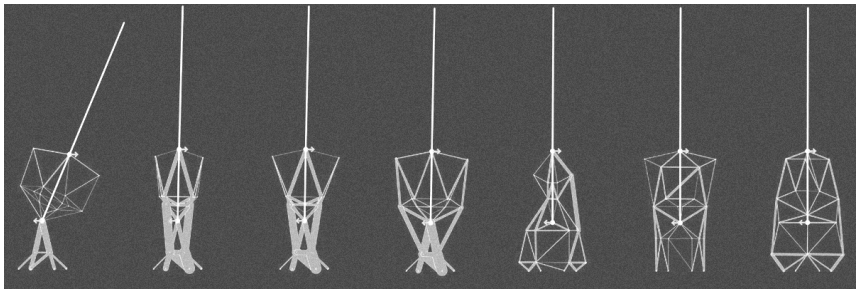
Helsinki Central Library, 2012

Project Information

- Internal Collaborators: Structures: Tom Foley, Duncan Horswill.
- External Collaborators: Alison Brooks Architects
- Location: Helsinki, Finland
- Project type: Public library
- Floor area: 10,000m^2
- Software/language used: Microstation Visual Basic, Java/Processing

Description of work

Finding principal stress lines for a random distribution of columns. These lines set out ribs on the soffit.



[27Cts]

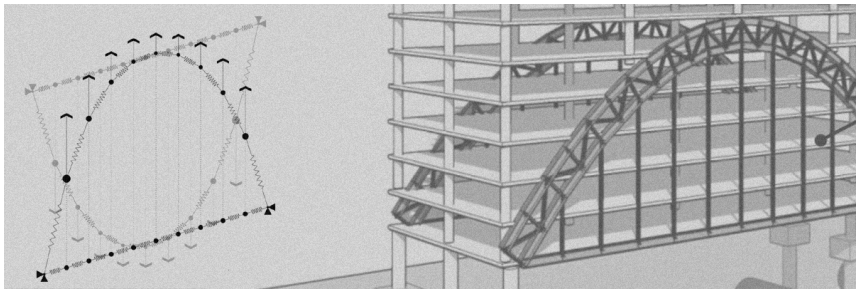
Citylife Tower Spire, 2012

Project Information

- Internal Collaborators: Structures: Mark Pniewski.
- External Collaborators: Isozaki & Associates Architects
- Location: Milan, Italy
- Project type: Tower spire
- Software/language used: Java/Processing

Description of work

The use of a genetic algorithm to generate designs for a tower spire. An example of using a metaheuristic and not a heuristic to incorporate other knowledge into the process. Deflection was chosen as the fitness criteria because a stress based approach rewarded mechanisms with the highest fitness. A Java application was written by the author that combined a modelling, structural analysis and a genetic algorithm.



[28Yor]

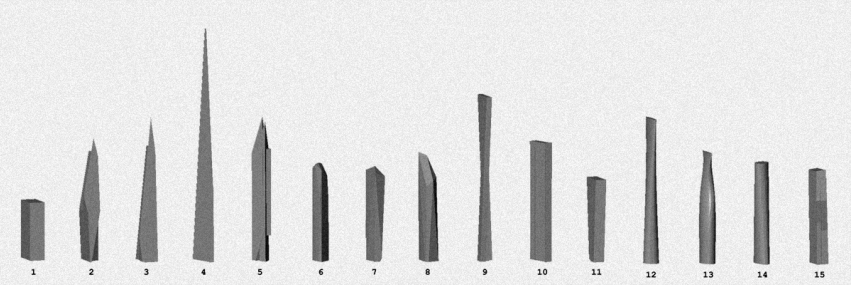
York Way Apartments, 2012

Project Information

- Internal Collaborators: Structures: Sebastian John Wood.
- External Collaborators: None
- Location: London, UK
- Project type: Residential
- Size: 25,000m^2
- Software/language used: Java/Processing

Description of work

Form-finding of a large two-dimensional arch structure with asymmetrical loads. Simple application of the dynamic mass method for finding an efficient arch shape.



[29Grt]

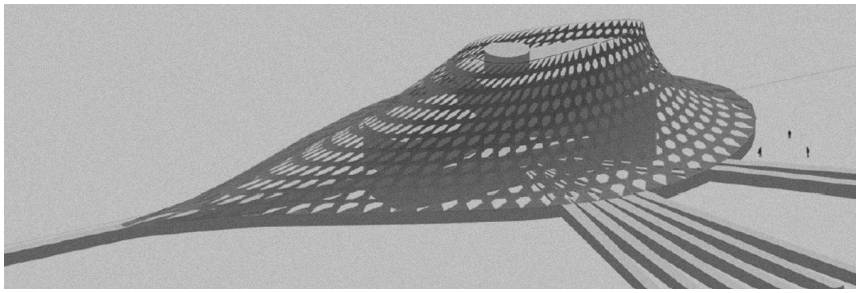
Gran Rubina Tower 3, 2012-13

Project Information

- Internal Collaborators: Structures: Duncan Horswill. Building Physics: Tom Shilton. Façades: Mark Pniewski.
- External Collaborators: AG5 Architects: Brian Sheldon, Daniel Nielson, Oliver Wong, Marc Wilson.
- Location: Jakarta, Indonesia
- Project type: Commercial/Office
- Gross external area: 36,000m^2
- Software/language used: Grasshopper + Embryo

Description of work

Design exploration incorporating a meta-parametric approach using Embryo. Early stage decision support feedback automatically was given to the design team for different tower designs of identical gross extenal area. The towers were produced using both manual methods and by using Embryo with the relative performance of each design compared.



[30Ess]

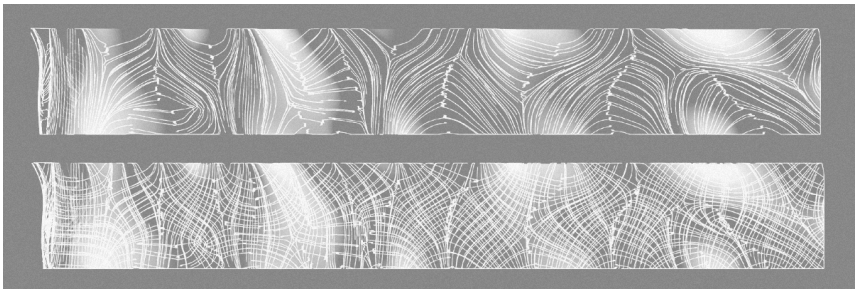
European Spallation Source (ESS) Building, 2013

Project Information

- Internal Collaborators: Structures: Rob Harrold. Façades: Mark Pniewski.
- External Collaborators: Foster + Partners
- Location: Lund, Sweden
- Project type: Research facility (invited competition)
- Floor area: 100,000m^2
- Software/language used: Java/Processing

Description of work

Support to Foster+Partners on the development of a large roof structure to house internal buildings. Form-finding software by the author was used to develop two different funicular designs, both of which were not progressed.



[31Tiv]

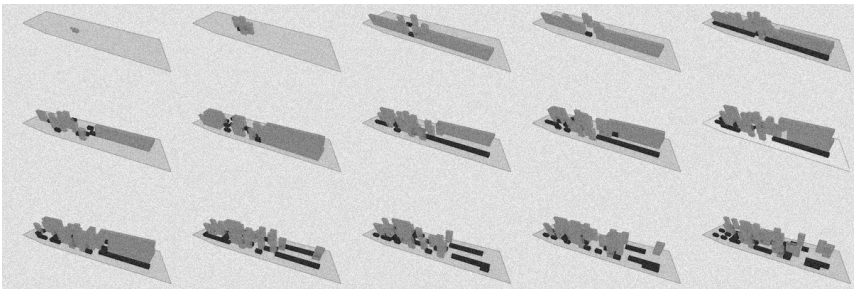
Tivoli Edge Development, 2013

Project Information

- Internal Collaborators: Structures: Duncan Horswill. Façades: Mark Pniewski.
- External Collaborators: (Ramboll Denmark)
- Location: Copenhagen, Denmark
- Project type: Retail/Other commercial
- Façade area: 1,800m^2
- Software/language used: Rhino Grasshopper

Description of work

Façade discretisation of a freeform doubly curved facade to a planar quadrilateral (PQ) mesh.



[32Plu]

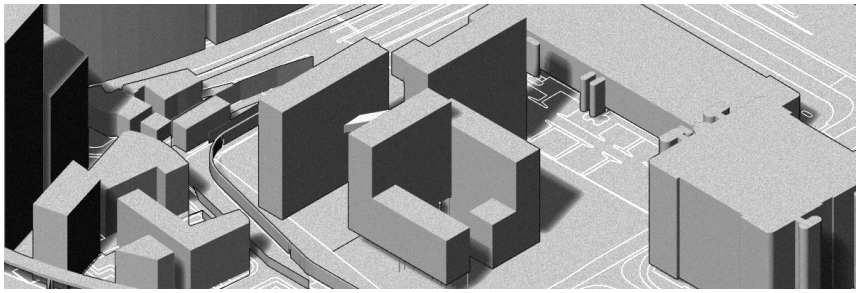
Pluit City, 2013

Project Information

- Internal Collaborators: Structures: Duncan Horswill. Infrastructure: Guy Collingwood.
- External Collaborators: SOM
- Location: Jakarta, Indonesia
- Project type: Urban Development
- Software/language used: Processing/Java

Description of work

Assisting the architect in generating urban layouts using low level rules. By using a 'class-II' cellular automaton, two-dimensional configurations met all local rule conditions when the system reached equilibrium. The complex nature of the process meant exactly how the configurations were being generated was practically impossible to follow.



[33Tow]

Tower Hamlets, 2014

Project Information

- Internal Collaborators: None.
- External Collaborators: 3DReid Architects: Charlie Whitaker, Antonios Lalos, Olympia Katsarou.
- Location: London, UK
- Project type: High density residential
- Software/language used: Grasshopper + Embryo

Description of work

Embryo was used to explore designs for a high density residential scheme. A formal grammar based on project-specific rules was constructed and combinations of parametric models explored. It was found that post-creation, the models could be understood and developed by the design team.

B. Principal Stress Trajectories

This section includes a short paper presented at the Ramboll Technical Forum in 2010.

Abstract

This paper presents an approach to the design of floor slabs, façades and roof structures based on principal stress directions for a continuum under self-weight. Examples of test projects are given including an installation design for the London Ramboll office foyer space. Bespoke software developed by Ramboll Computational Design (RCD) is shown which creates principal stress fields in real-time in order to present constant feedback to the designer as boundary conditions are adjusted. In this sense, structural performance is set as a constraint before the initial concept design stage has begun, and an afterthought once the geometry is frozen.

The work presented here was conducted with assistance from the Ramboll Computational Design team, namely Tom Foley and Stephen Melville. John is studying for an Engineering Doctorate (EngD) at the University of Bath in computational design with academic supervisors Paul Shepherd and Chris Williams. The work is supported by the EPSRC and Ramboll.

Figure B.1: Concrete slab (Viikki Synergy Building [10Vik]) with ribs aligned with principal bending stress paths

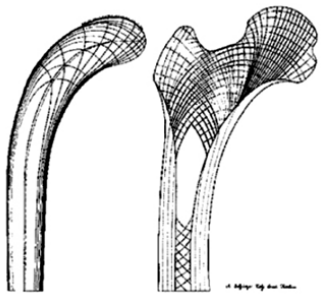


Figure B.2: Wolff's sketch of the femur showing material alignment

B.1. Introduction

As early as 1800s, biomedical engineers Culmann & Von Meyer (1867) had proposed the 'trajectorial theory of trabecular bone structure.' This analysis showed that material appeared to be aligned along the principal lines of force going through the bone allowing for high efficiency of material whilst keeping a single bone as lightweight as possible overall. Following on from this work, the German anatomist Julius Wolff proposed that not only that trabeculae were aligned along principal stress lines (fig.B.2), but that these orientations could adapt dynamically over time should the bone be used in a different way.

The work of the engineer/architect Pier Luigi Nervi took inspiration from such natural processes and explored the idea of minimising material usage by aligning concrete ribs along lines of force, notably in his Gatti Wool Factory project whereby lines of principal bending stress were set with deep ribs, all within a fully repeatable unit (Nervi, 1965). The Swiss Engineer Heinz Isler also found that reinforcing concrete shells along the lines of principal stress was a very good way to improve efficiency by minimising the amount of steel used, and hence reduce additional weight (Gruber, 2011).

Modern day research in this field includes the work of Kaijima & Michalatos (2007) and Winslow et al. (2007) whereby aligning material to principal stress lines on a pre-determined free-form surface shell has been found to have structural benefit.

B.2. Real-time Software

Following this research it was decided that RCD should investigate the benefits of this approach. Rather than use standard analysis software for a predetermined problem, software was developed that finds a principal stress vector field in real-time for a given set of adjustable boundary conditions. Ribs or members (dependant on the design problem) are then

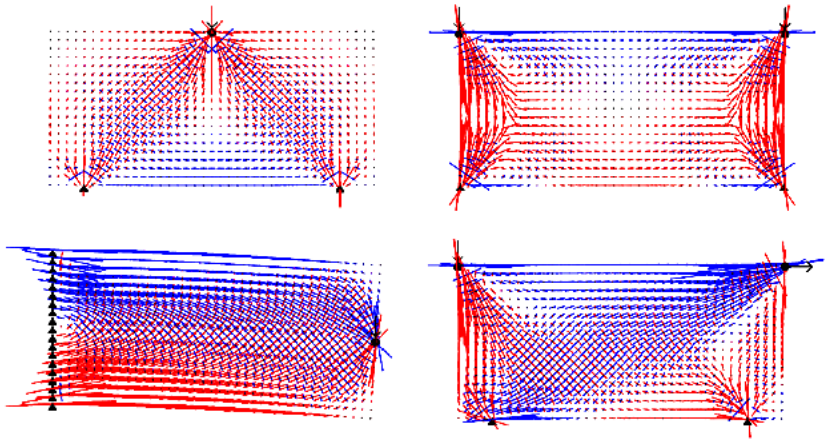


Figure B.3: Screen-shots from the software for solving plane stress problems in real-time

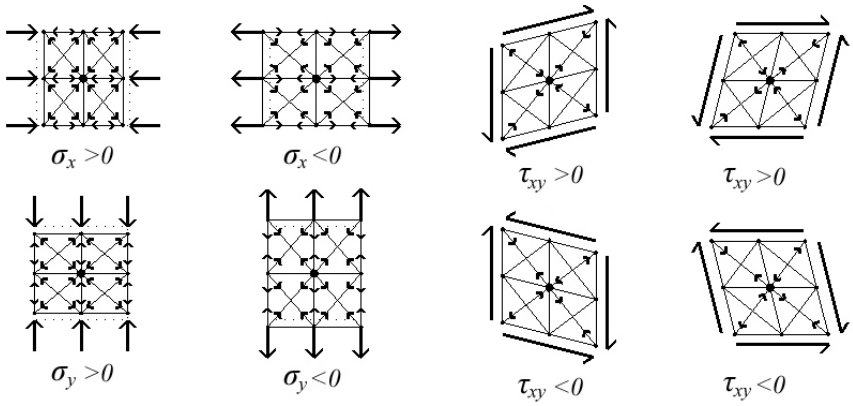


Figure B.4: Spring finite element

orientated along the principal stress field using a streamline integrator with a seeding method to ensure equal density (Mebarki *et al.*, 2005).

Different streamline patterns can be explored with their aesthetic qualities assessed before design freeze. This is due to the software's ability to calculate the stress field rapidly using a form of the dynamic relaxation method (Day, 1965). This speed means the designer is able to investigate the effect of different loads, supports, boundaries, etc. and receive real-time feedback to better guide the placement of material along sound structural principles. The simulation does not need to be calculated from scratch each time like traditional FE methods should boundary conditions change.

B.3. Calculation Method

The software uses a special type of element made from a braced rectangular framework of springs (fig.B.4) which make up the plane. The number of rectangles used depends on the accuracy required. From this orthogonal base, principal directions and magnitudes are derived using the values of normal and shear stress for each node and its adjoining

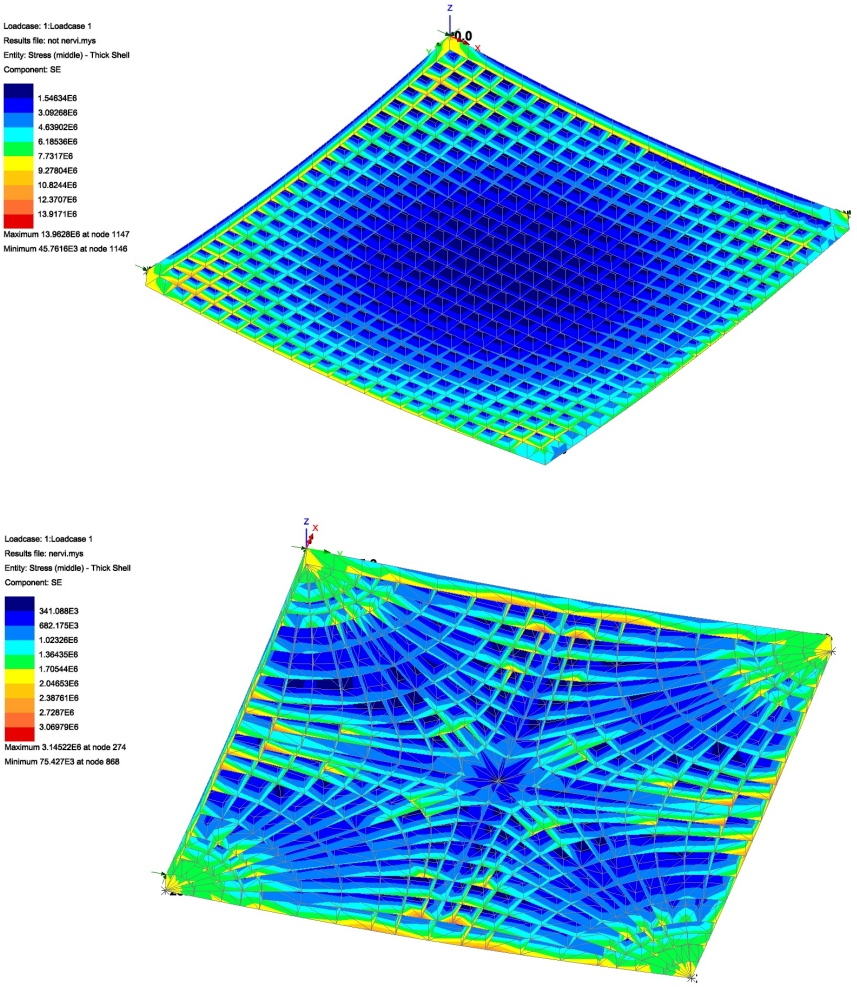


Figure B.5: Identical volume of concrete and rib depth for waffle and 'Nervi' slab leading to a large deflection reduction at centre and better stress distribution

springs. Normal stresses are calculated using the rectangular spring grid whereas shear uses the diagonal bracing springs. The streamlines generated from the calculated stress field are orthogonal and known as conjugate curve networks, meaning intersections are always perpendicular. This gives benefits in terms of fabrication especially for roof designs made of discrete elements, but also for the design of concrete formwork as there are no acute angles.

B.4. Principal Bending Stress

Aligning material along the principal bending stress lines was proposed for use on the Viikki Synergy Building [10Vik]. By increasing the overall depth slightly to 50mm and removing material, there was found to be a 33% concrete saving and a small reduction in the required reinforcement (tableB.1) for a similar deflection.

A early design for the London office foyer project [07Foy] was also proposed using lines of principal bending stress (fig.B.6).

Property	Flat slab	Optimised slab
Maximum depth (mm)	300	350
Concrete Volume (m3)	182	120
Reinforcement weight (kg)	6450	6400

Table B.1: Viikki typical slab comparison

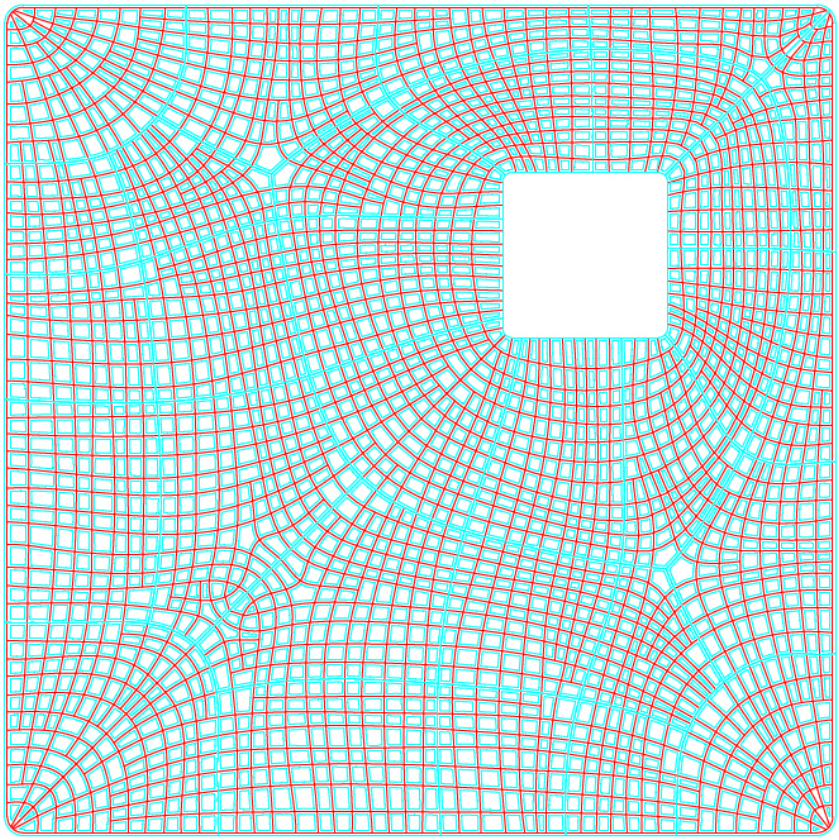


Figure B.6: Ribbed slab design for foyer installation with simply supported outer boundary perimeter and inner void

B.5. Conclusion

This paper outlines a simple method of aligning material along principal stress lines and introduced a real-time software application to explore designs. Clearly, the fabrication of such designs is more complex than traditional corrugated or waffle slabs and hence the efficiency savings will need to be measured against these by talking to specialist contractors.

C. Structural Form-finding using Springs and Dynamic Weights

This section describes the dynamic mass method for real-time funicular form-finding. The trivalent network approach to finding three-dimensional shell structures (Section C.4) builds on work by the author first published at the International Association for Shell and Spatial Structures Annual Symposium in 2011 (Harding & Shepherd, 2011).

The approach can be implemented in two or three dimensions. The resulting found geometry consists of purely axial forces under self-weight, with zero bending moment at nodes for both shells and tension net forms. A real-time dynamic relaxation solver is used to achieve static equilibrium. By using a relaxation method, the designer is able to alter the gravitational field or apply new point loads without re-starting the analysis, thus leading to a more interactive experience during design exploration.

C.1. Background

The form-finding of funicular structures has long been advantageous for designers of compression shells and tension nets due to their zero out-of-plane bending moment property under self-weight. There are many examples from the past of attempts by designers to generate such efficient funicular forms within certain boundary constraints including both physical and computational models or combinations of both. Antoní Gaudi's physical hanging chain models have inspired many engineers to use similar methods in funicular form-finding including Heinz Isler (Chilton, 2010), Felix Candela (Faber, 1963) and Frei Otto (1995) to name but a few. Such methods give a real-life understanding of the behaviour of material subject to self-weight but at the cost of having to model each design option individually which can be extremely time consuming and constraining, especially if the boundary conditions are complex or the exact requirements of the design are not known.

Recent advances in computing power have made design exploration possible using direct simulation. Active statics (Greenwold *et al.*, 2003)

for example is a development of Culmann's graphic statics (1875) that allows the user to interact with the system and directly see the feedback from various actions, although it is limited to solving problems in two dimensions. In three dimensions, Killian and Ochsendorf (2005) have employed particle spring systems with stiff springs in order to approximate rigid links. By using a real-time solver, the designer is able to explore different funicular designs quickly by changing boundary conditions and adding or subtracting additional links. Such software aims to *mimic* simulations of natural systems such as Gaudi's hanging chain models. More recently Block and Ochsendorf (2007) have developed Thrust-Network Analysis (TNA) using linear optimisation in order to explore funicular forms in real-time, in particular vaulted masonry structures.

C.1.1. Zero-length springs

In 1932, the physicist Lucien LaCoste (1935) first discovered zero-length springs and applied them to the design of seismographs and gravimeters. The force exerted by such a spring is exactly proportional its length which is also its extension. As length is always positive, so zero-length springs are always in tension. This paper explains how zero-length springs may be used in particle spring system in order to form-find funicular structures by varying the lumped nodal weight applied at each node in real-time.

C.2. Two-dimensional systems

C.2.1. Method overview

A simple particle-spring system is constructed with nodes joined with zero-length springs (fig.C.1). An equal mass is lumped at each node and stays constant for each node in the system.

The springs are equally stressed at the initial condition (a) with nodes equally spaced along the x-axis. The property of zero-length springs mean that we then need only solve for the y component of the force when a gravitational field is applied in the negative y direction. For each node we therefore have the residual vertical force in y given as:

$$F_i^V = k \cdot (y_{i-1} + y_{i+1} - 2y_i) - mg \tag{C.1}$$

where y_{i+1} and y_{i-1} are the coordinates of adjacent nodes, k is a global stiffness constant equal for all the springs in the system and g is the

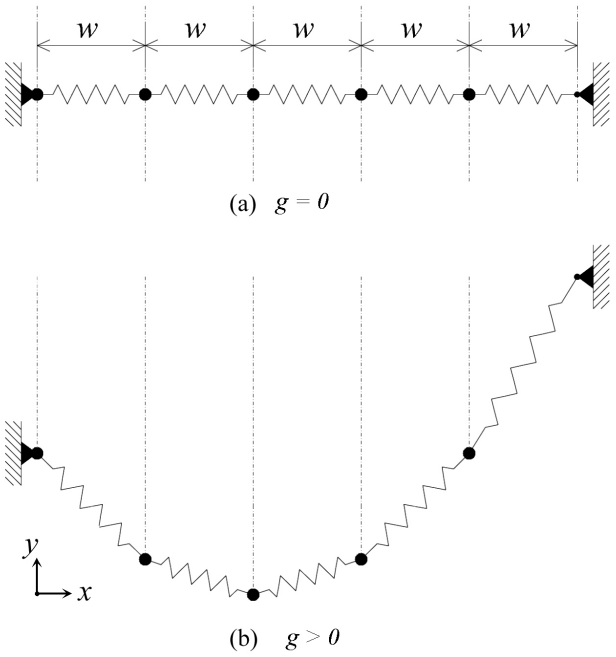


Figure C.1: Zero-length springs form a parabola with equal masses

gravitational constant. The system of nodes can be solved iteratively using a numerical method such as dynamic relaxation (Day, 1965) in order to reach static equilibrium. As the system is statically determinate, upon reaching equilibrium the springs may be replaced with idealised rigid bars with the pin joints remaining. The forces in these bars will be identical to the springs, and the system remains in static equilibrium for resisting the loads. As the zero-length springs are always tensile, so the bars will all be in tension. Consequently, reversing the gravitational field gives compression-only funicular forms.

C.3. Extension to funicular forms

C.3.1. Dynamic Weights

By making the lumped mass applied at the node proportional to the length of the adjoining zero-length springs, we essentially retain the same mass per-unit length for the springs - much like a catenary cable. This mass in turn affects the length of the spring itself and hence a coupled relationship is formed between the two. Indeed, the mass term can now be substituted, giving the following non-linear equation:

$$F_i^V = k \cdot (y_{i-1} + y_{i+1} - 2y_i) - \frac{g \cdot (l_1 + l_2)}{2} \tag{C.2}$$

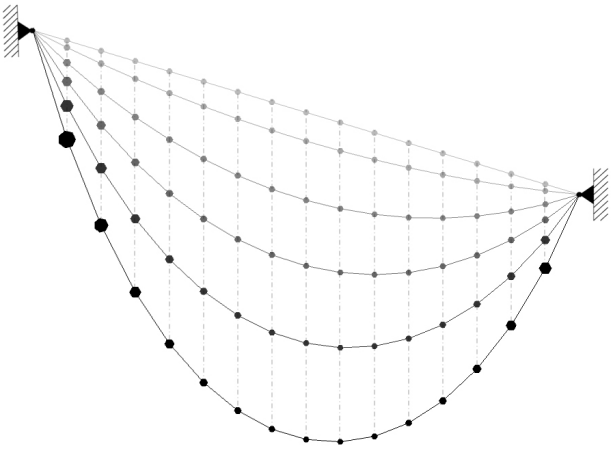


Figure C.2: Exploring catenaries by varying the gravitational field

$$l_1 = \sqrt{(y_{i-1} - y_i)^2 + (x_{i-1} - x_i)^2} \tag{C.3}$$

$$l_2 = \sqrt{(y_{i+1} - y_i)^2 + (x_{i+1} - x_i)^2} \tag{C.4}$$

The spring system can be solved numerically as before. The mass term will overpower the spring forces should g > k as the nodes continue to accumulate mass too quickly for the springs to resist the extra force. Even if this condition is met, the system can also be too stiff and not converge. In practice this is rectified by choosing a stiffness constant appropriate for the time-step used. This is easily done manually by visual inspection.

Again, we may replace the springs with bars due to the static determinacy if the system. This time, the nodal weights balanced are based on the weight of the bars themselves (assuming an equal mass per unit length). Again, as the springs were in tension, so the forces in the bars will be identical and compression-only if the gravitational field is reversed.

C.3.2. Real-time exploration

Various forms may now be explored within the two pinned boundary conditions simply by varying the gravitational field. Figure C.2 shows different length discretised catenaries formed quickly in real-time with the user varying the gravitational constant.

C.3.3. Catenary Comparison

An analytical solution to the continuous catenary in two dimensions is well known and is given by the hyperbolic cosine function. The method

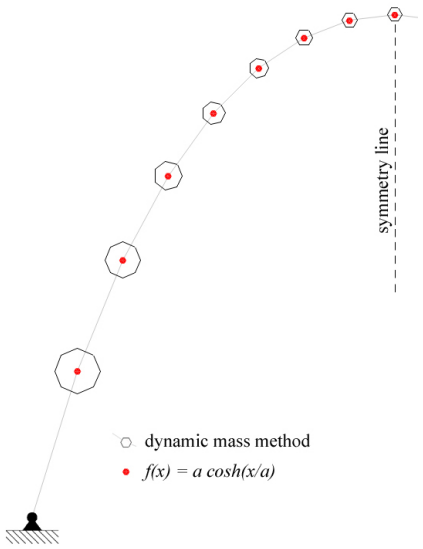


Figure C.3: 17 node system compared to an actual catenary

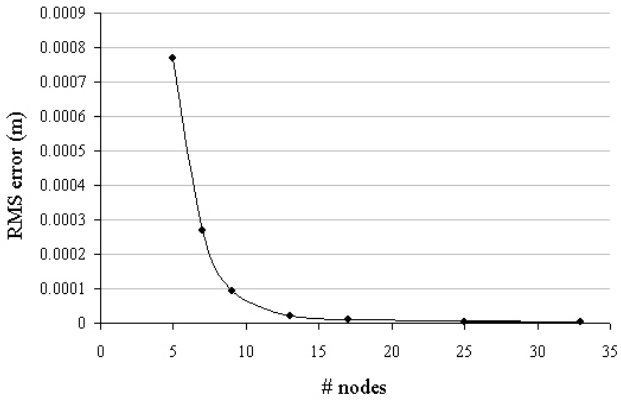


Figure C.4: Increasing of nodes reduces error to zero

generates accurate funicular forms for discretised catenaries with straight members. By increasing the number of nodes in the system we can very closely approximate a continuous catenary with even a relatively small amount of nodes.

Figure C.3 shows the form-finding of a 16m high catenary arch with nodes at unit spacings along the x-axis. With this problem, the maximum error was found to be only 3mm where the curvature of the catenary was greatest. By increasing nodes this error converges monotonically to zero (fig.C.4).

C.4. Three-dimensional systems

So far approaches to 2D problems have been investigated and hence have no additional value in terms of finding form. However, the method applied to 3D systems has much more use for the design of funicular shells in compression or tension. As has been discussed, transferring

from 2D to the complexity of doubly curved structures embedded in 3d space can be challenging for some other approaches.

C.4.1. Using a Trivalent Topology

By using a hexagonal starting mesh (fig.C.5), statically determinate systems can be solved for equilibrium in three dimensions and hence unique funicular solutions found. The lumped mass applied at the node is related to either the three adjoining springs (for discrete systems) or adjoining areas (for continuous shell form-finding). It is known that a 3d system in equilibrium will also be so as a 2d system when projected to any plane embedded in 3d space (Henrici & Turner, 1903). As the gravitational force becomes zero when projected to the ground plane for every node, it follows that the system is statically determinate for 3-valance nodes. This property is also used by Thrust Network Analysis (Block & Ochsendorf, 2007).

This determinacy is important for this method because if the springs in the system are replaced with rigid bars, the system is a unique solution for the given masses that now reflect the weight of the bar or areas elements. In addition, the determinacy enables the designer to view the final axial member forces in real time whilst relaxation is taking place, and as loads as boundary conditions are modified.

Instead of dynamic weights relating to the length of the springs, area loads may be applied in order to find continuous shell structure thrust networks as well as discrete ones. Appropriate areas are easily found by using the triangular dual network from the hexagonal topology. The zero-length springs mean that a smooth shape always results, reducing discontinuities in the final shell form.

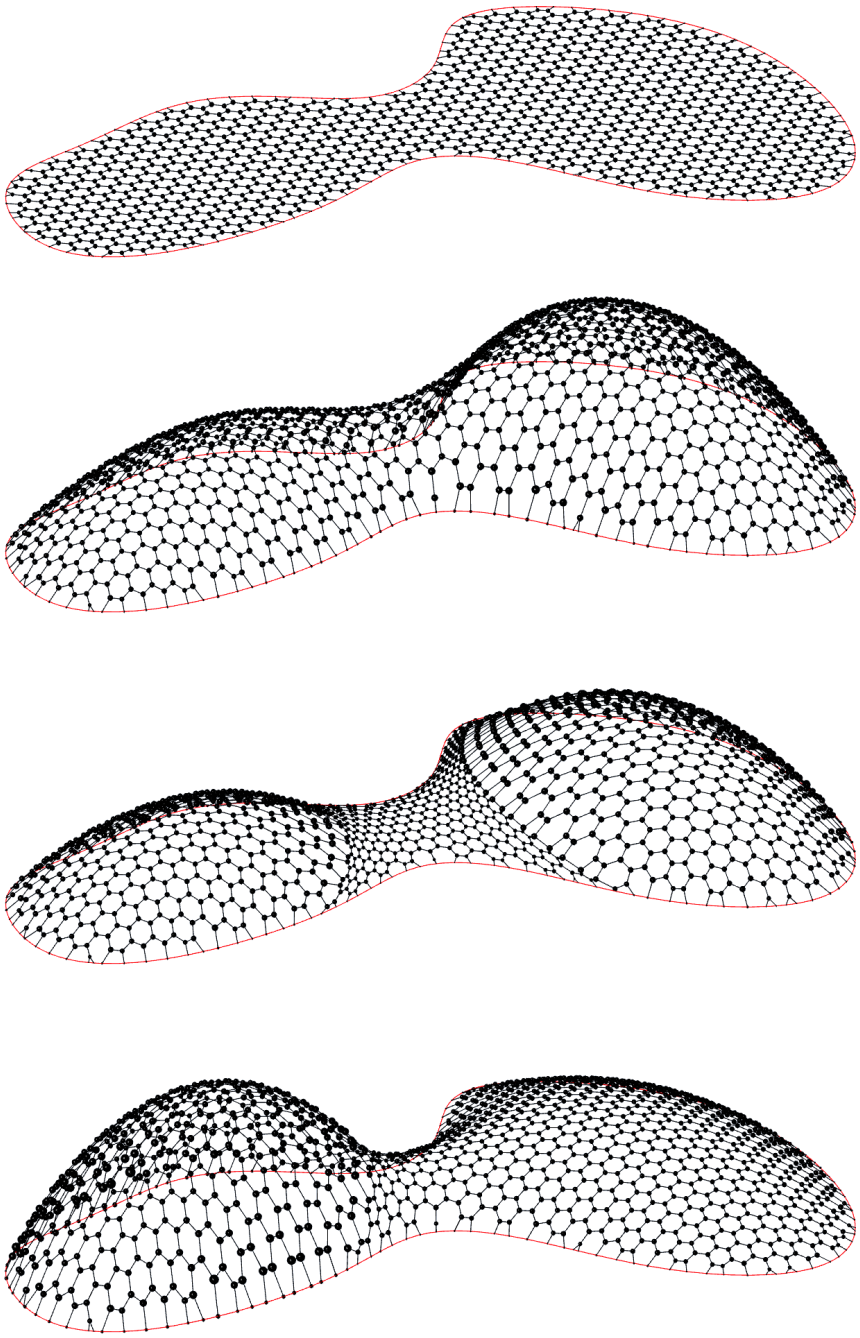


Figure C.5: Trivalent spring network. Varying spring stiffnesses gives different results that are still funicular

D. The London Foyer Sculpture

D.1. Introduction

In April 2011 a plywood gridshell sculpture was constructed at the Ramboll UK main office in London [07Foy]. Located in the entrance foyer space, the installation was designed to implement and evaluate some of the recent research conducted by the newly launched Ramboll Computational Design (RCD) group. The foyer sculpture is designed for both structural and fabrication considerations simultaneously. This makes it conceptually different to the current trend of architectural free-form surface modelling where structural and fabrication performance is post-rationalised following concept design.

D.2. Design Strategy

The first step was to find a funicular compression shell to sit perfectly within the confined boundary conditions and allow the current use of the space as an entrance foyer to be uninterrupted. This was achieved using in-house software (Harding & Shepherd, 2011), allowing the design team to explore various design options in real-time (fig.3.2.1).

Following the form-finding stage a member discretisation of the doubly curved form was undertaken. Members were aligned along the principal curvature network which was also found to be similar the principal stress field for a continuous shell. Such a strategy thus resulted in structural efficiency as well as being simple to fabricate from standard sized timber sheets.

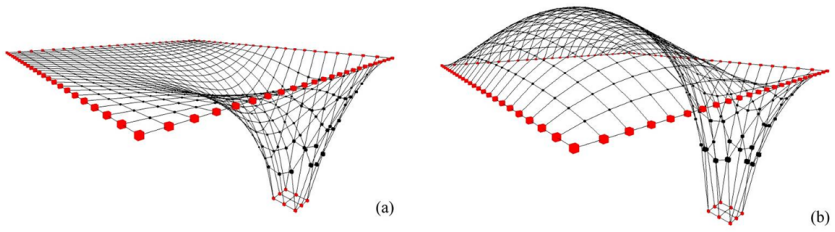


Figure D.1: Funicular form-finding approach

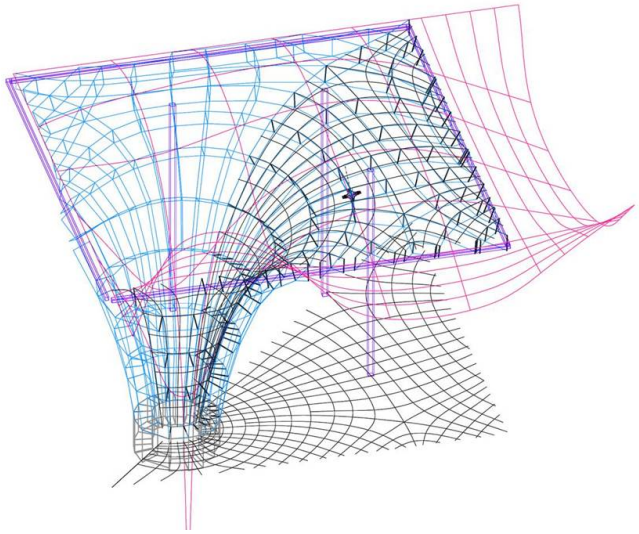


Figure D.2: Members set out following surface principal curvature field

D.3. Material & Fabrication

All members in the shell were laser cut from standard sized flat pieces of 6mm FSC sourced Malaysian red hardwood WBP exterior plywood. The type of wood was chosen due to its low cost, appearance, sustainability credentials, compressive strength and ease of fabrication with a laser cutter.

Due to the principal curvature orientation of the members there was zero twist required along their length in order to form the doubly curved shell (fig.D.2). As the members are also aligned with the principal stress field the self-weight of the structure travels mostly in axial compression to the supports. The density of the member discretisation is kept constant allowing the self-weight to be similar to that of the continuous shell.

To achieve the desired form, although flat, each member was unique and hence a laser cutter was used to cut the standard sheets of plywood. Due to each member being a constant depth, efficient nesting of the elements on each sheet was a simple task, allowing for minimal material wastage during manufacture. Reference numbers were scorched onto the members during the cutting process to assist with assembly.

D.4. Connection Details

Every node connection is also laser cut plywood, connecting adjoining members and transferring load using a single bolt normal to the surface continuum. By locating nodes on the funicular form, the bending moment is virtually zero allowing for a small size and reduced material

Figure D.3: Foyer sculpture connection detail

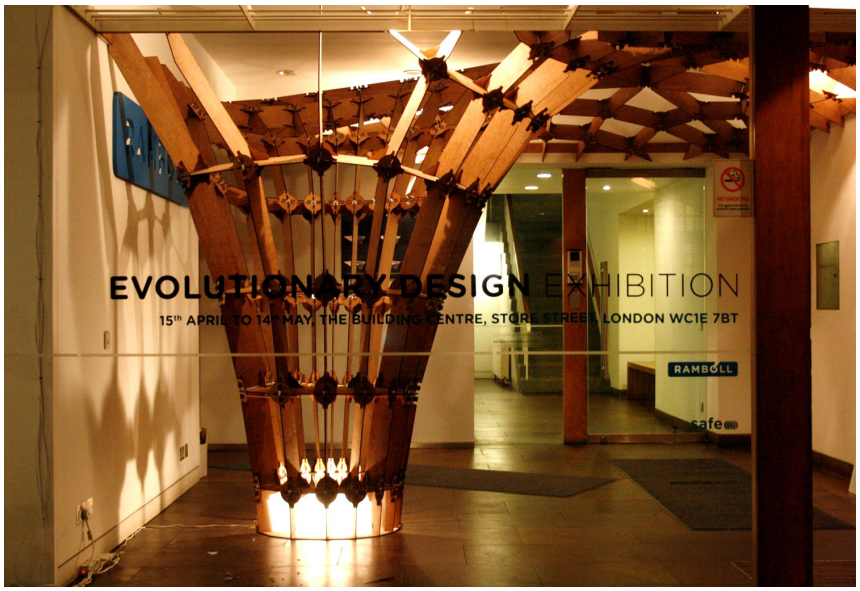


Figure D.4: Final constructed gridshell

usage. In theory, the bolt is not required under self-weight, but are located purely for accidental load. Similar to the members, the node connections are numerically referenced using the laser to make assembly on site straightforward (fig.D.3).

D.5. Assembly

The project was constructed in two days with the final result shown in fig.D.4. The whole structure is able to be unbolted, flat packed and transported using only a medium sized car, thus meaning the sculpture will have a life beyond that of its existing home at the Ramboll office.

E. The TRADA Pavilion – A Timber Plate Funicular Shell

This section is a preprint of a paper co-authored by the author and presented at the International Association for Shell and Spatial Structures Annual Symposium in 2013 (Harding & Lewis, 2013). It describes the process used on the TRADA pavilion project by Ramboll Computational Design.

Abstract

This paper describes the design and construction of the TRADA (Timber Research and Development Association) Pavilion by Ramboll Computational Design. The design process combined a zero-length spring funicular form-finding approach with a planar polygon discretisation method, thus enabling the final structurally efficient but complex geometric form to be realised with low-cost materials. The timber plate shell is fully demountable and has been assembled at several separate locations since being completed in September 2012. To the authors' knowledge, the structure is one of the largest free-form faceted thin shell structures ever to be constructed. It is hoped that the project inspires a new approach in realising complex funicular shapes with low-cost materials and fabrication methods

E.1. Introduction

The TRADA Pavilion was designed and first constructed by the Ramboll Computational Design (RCD) team in 2012. The structure is a doubly curved compressive shell assembled entirely from flat timber panels and stainless steel hinges. Designed as temporary trade fair stand, the shell is fully demountable. This allows it to be used at various trade events across the UK, promoting the use of timber as a building material. At the time of writing, it has been assembled at three different locations. The design of the pavilion was separated into two main stages. Firstly,

the form-finding of a funicular compression-only shell was undertaken. The form-finding approach chosen was a zero-length spring system with dynamic nodal masses. Its use is explained in Section 2 of the paper. Secondly, a surface discretisation into planar polygons with tri-valent nodes was made in order to cheaply and efficiently realise the doubly curved form with flat timber panels. This discretisation had to be sufficiently fine in order to maintain a similar structural behaviour to the continuous shell. This process is covered in Section 3 of the paper. This combination of techniques meant that a structurally efficient compressive shell with a complex form could be realised using low-cost off-the-shelf materials and well known fabrication methods, as well as providing a demountable structure as required by TRADA.

E.2. Funicular Form-finding

The form-finding of funicular structures has long been desirable for compression and tension shells. Antoní Gaudi's original physical hanging chain models have inspired many designers to explore similar methods for finding efficient shells, with notable work by the engineers Isler, Candella, Torroja, Nervi and Otto to name only a few. Physical hanging nets working in tension-only under self-weight are reversed to act in pure compression under the same loading, a principle first discovered by Robert Hooke in the 17th century. As the physical tension nets cannot carry bending, the resulting compressive forms that result also have zero-bending under self-weight, with the structure working axially with no out of plane forces thus enabling a thin structural depth for the shell.

E.2.1. Computational Approach

Whilst using physical form-finding models give a real-life understanding of the behaviour of material subject under self-weight, the cost of having to model many design options individually can be both time consuming and constraining if the problem's boundary conditions are complex and/or the exact design requirements are not known a priori. In response to these difficulties, recent efforts have been made to recreate the playfulness and intuition of 3D physical model interaction in computer software applications. Interactive software examples include directly mimicking physical models with stiff springs (Kilian & Ochsendorf, 2005), as well as the more recent 'Thrust-Network Analysis' (TNA) method that combines linear optimization with projective geometry and duality theory (Block & Ochsendorf, 2007).

E.2.2. Zero-Length Spring System

The computational method used for the TRADA Pavilion utilises a zero-length spring approach with dynamic weights (Harding & Shepherd, 2011). Zero-length springs by their nature must always be in tension, and hence any resulting form in tension will result in a compression only-structure when used appropriately during structural form-finding. An initial hexagonal zero-length spring system was established to the dimensions of the 8m x 6m site. Each node thus consisted of 3 springs meeting at a pinned connection. Nodal mass forces were then applied perpendicular to a reference ground plane, with equilibrium solved for using the dynamic relaxation method. With such an approach, funicular forms were able to be generated so long as the lumped nodal masses applied are updated at each time-step.

As the TRADA shell was to be a continuous surface as opposed to a lattice, the lumped mass applied at each node was proportional to the local surface area. This proportionality assumed that the material thickness was constant throughout, although exactly what thickness did not need to be known at this stage. The local area was easily approximated by using the hexagonal mesh dual which is a triangle. Within a gravitational field, the mass therefore exerts a point load perpendicular to the ground plane in an upwards direction, whose magnitude is proportional to this local area. This force is therefore resisted by the adjoining springs.

Because this applied nodal force vanishes to zero when projected to the ground plane, the system is statically determinate for 3-valance nodes, so long as all of the applied nodal forces are indeed perpendicular to the ground plane. This condition is true for the form-finding of funicular structures where self-weight is the dominant load case, a fact also utilised by Thrust Network Analysis (Block & Ochsendorf, 2007) when unique solutions are sought, inspired by the work of Williams (1986) on reciprocal structures.

The static determinacy of the hexagonal system is useful during the form-finding process, as it enables the designer to view local forces in real time whilst the model is altered and new designs are explored. If the springs are replaced with rigid bars after equilibrium is found, the forces in the elements must remain as per the spring model in order to resist the applied nodal point loads due to the uniqueness of the solution. By reversing the direction of the point loads after finding a tensile form, a purely compressive shell with zero-bending is the result (fig.E.1).

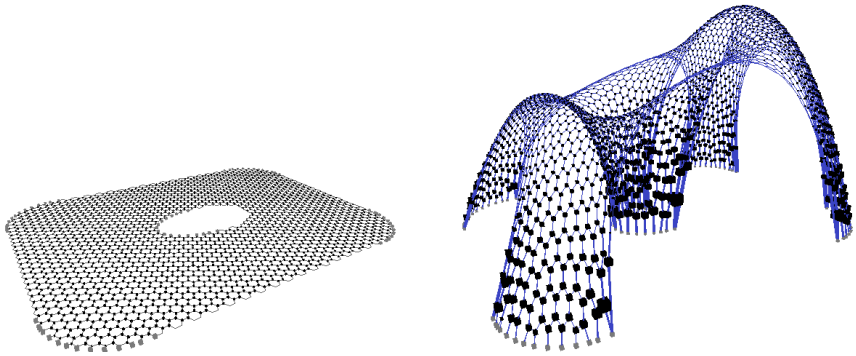


Figure E.1: Shell form-finding using the dynamic weights

E.2.3. Software Application

By incorporating the form-finding method into a Java application, spring stiffnesses, the gravitational constant and boundary conditions could be adjusted on the fly during design development. Various funicular forms could therefore explored during the early design phase. This included the introduction of a negatively curved area at the centre of the site, and the four free edges around the perimeter of the shell. By using zero-length springs, not only were all elements guaranteed to be in tension, but also a much smoother shell resulted than was the case by using natural lengths set by the initial line lengths with no gravitational field applied, i.e. on the ground plane.

A real-time approach meant that the effect on curvature could be quickly assessed by changing the boundary conditions. For example, one of the main problems of the design was to ensure a sufficient amount of double curvature was present in the 4 corner legs to counter buckling effects. However, the support areas were limited by their size as the structure had to be accessed by the public. By adjusting the size and shape of the legs as they meet the ground, the overall effect on the shape could be assessed quickly until a satisfactory compromise was found. Once the form was finalised, rather than having to painstakingly measure a complex physical model, a fine triangulated mesh could be exported ready for the next stage of the process (fig.E.2).

E.3. Planar Re-meshing

Due to their double curvature, compressive shell structures have traditionally been constructed from concrete or masonry. For the former, bespoke formwork is required in order to accurately provide a reference surface before the pour is made. Reinforcement is also necessary to resist live load cases due to concrete's poor performance in tension, and must follow the curvature of the shell - often a complex and time-consuming

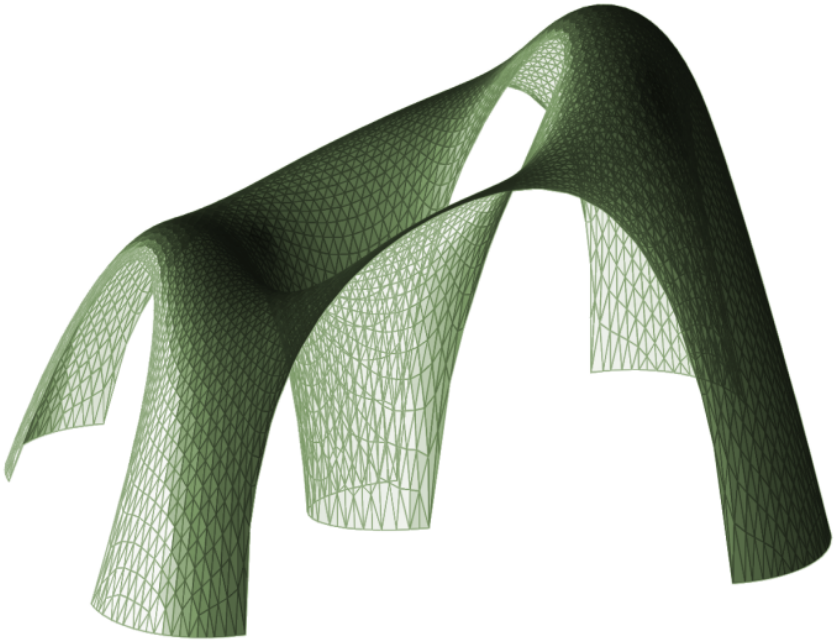


Figure E.2: Final fine triangulated mesh sent for re-meshing

process. Such cost implications have seen the popularity in thin concrete shell structures fall in recent decades. Timber on the other hand, works well in tension and compression if cross-laminated, and hence as a shell material does not require extra reinforcement. However, forming doubly curved timber surfaces becomes prohibitively expensive as each panel would have required a unique jig or be highly wasteful of material. A suitable compromise therefore was to discretise the shell into flat panels of timber, whilst remaining close to the original surface so as not to introduce significant bending moments into the structure. This approach also allowed the structure to be demountable and flat packed for easy transportation.

E.3.1. Triangles & Quads

The simplest planar discretisation possible is to form triangular panels from the initial surface mesh. Nodes are 6-valent, that is, 6 edges are incident per node. This was the first method attempted (fig.E.3). However, although the result follows the form closely, there were a number of structural concerns about adopting this solution.

The triangulated mesh was found to have lines of edge continuity throughout the surface which allowed the shell to fold as it deforms under both dead and live load cases, a situation not helped by the free edges. It therefore required stiff connections to resist this bending action. Discretising the shell into planar quads with 4-valent nodes by aligning edges to the principal curvature of the surface (Liu *et al.*, 2006) was also ruled out for a similar reason.

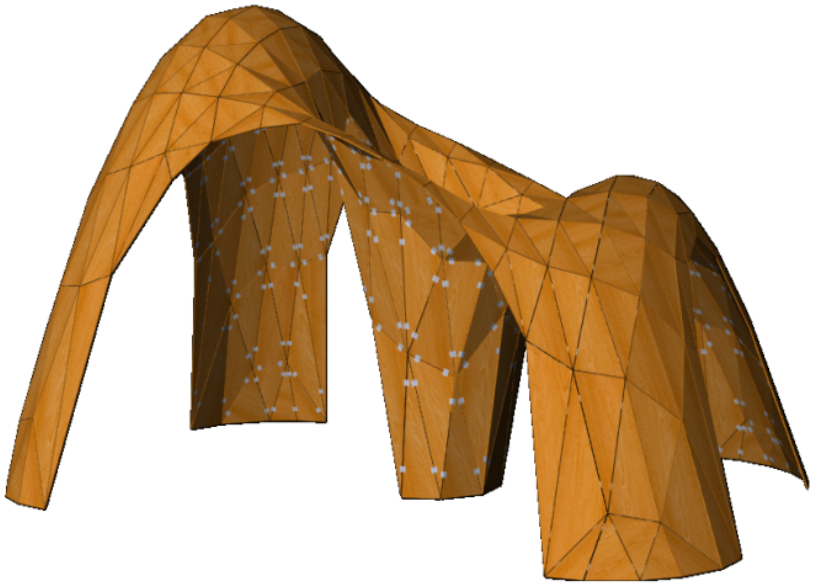


Figure E.3: The funicular surface discretised with triangular plates

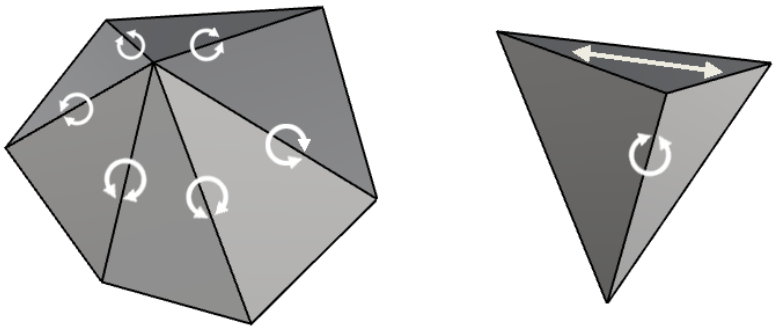


Figure E.4: The three-plate principle. Unlike a 6-valent triangular mesh, a 3-valent planar mesh requires no edge restraint in order to remain locally stable

By using a trivalent node however, local stability can be maintained even with an entirely hinged edge that offers no bending resistance, (fig.E.4). Any loads applied out of plane are taken via torsional restraint along each edge, i.e. where two plates meet. Ture Wester incorporated this property of 3-valent plate shells in his plate-lattice dualism approach (Wester, 1992). Less edges incident per node also led to neater details over the whole structure and perhaps visually a more interesting result.

Hexagonal 3-valent plate structures that utilise this property can be found in nature, for example, the tessellation of plates on a sea urchin (Wester, 2002) and the formation of 'scutes' on turtle shells. Interestingly, although this approach also uses the underlying structural properties of a 3-valent system as with the form-finding stage, for the planar re-meshing it does so in a completely different way.

The benefits of a 3-valent plate system have also been investigated extensively by Bagger et al. in relation to structural glass shells (Bagger, 2010). With triangular plates (6-valency), load concentrates in the edges of the elements whereas with a 3-valent system, load is transferred through the structure via in-plane stresses, thus acting similarly to a continuous shell. This better distribution of stress therefore allows for a thinner material to be used for the plates themselves.

E.3.2. Tri-valent Planar Polygons

Discretising a doubly curved surface into 3-valent plates has only recently been achieved. Cutler and Whiting (Cutler & Whiting, 2007) were the first to adapt a technique developed in the computer graphics industry known as Variational Shape Approximation (Cohen-Steiner *et al.*, 2004) for use in the architecture by integrating the approach with intersecting planes. Around the same time, Troche (Wang *et al.*, 2008) independently developed a similar Tangent Plane Intersection (TPI) approach specifically for hexagons. More recently, Zimmer et al. have investigated generating dual supporting structures for flat plates using Variational Tangent Plane Intersection (Zimmer *et al.*, 2013).

Using the same approach as Cutler and Whiting, the RCD team developed their own software in C# for use in Rhino Grasshopper. This meant that discretisations of varying densities could be quickly assessed in terms of assembly time, fabrication costs, overall appearance and structural behaviour, i.e. its deviation from the original smooth form.

The final discretised mesh consisted of 152 flat panels, each of which was unique. Although the appearance was irregular, each panel's shape was related to the curvature of the surface, see fig.E.5. In areas of positive Gaussian curvature, such as the dome-like areas at the top of the structure, the panels are convex. In areas that are close to cylindrical or have approximated zero Gaussian curvature, such as the legs, the panels are generally rectangular. Finally, in areas of negative curvature such as the central funnel, interesting concave "bow-tie" shapes are present.

E.3.3. Free Edges

Although the tri-valent mesh provided sufficient restraint for the internal shapes, the free edges however were not sufficiently restrained and hence vulnerable to buckling. Perpendicular edge stiffeners were therefore required to ensure stability, similar to the upturned edges used by Heinz Isler on his concrete shells (Chilton, 2000).

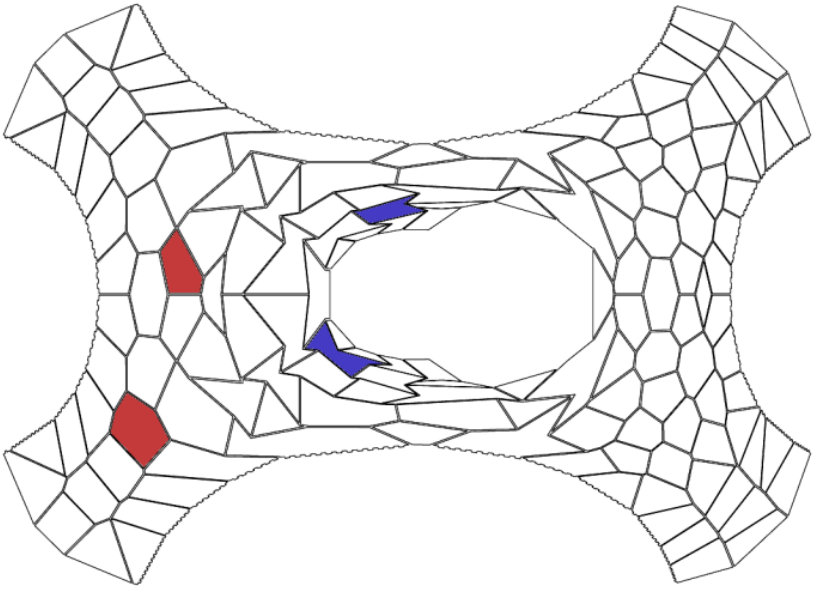


Figure E.5: Plan view of the final discretisation. Each panel's shape reflects the underlying curvature of the surface at that location.

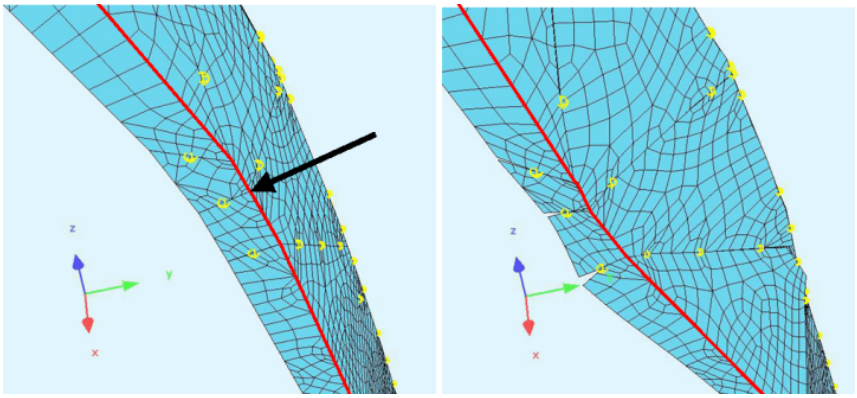


Figure E.6: Edge of the structure resisting accidental point load through an edge stiffener at a free edge (exaggerated deflection shown)

Non-linear analysis of the structure was conducted to validate the approach and ensure that the same plate thickness could be used as per the main shell (fig.E.6). This analysis was combined with full-scale physical testing of a leg before committing to the design.

E.4. Fabrication & Assembly

By discretising the shell form into planar polygons the shape could then be realised with flat timber panels. 15mm thick birch plywood was chosen due to its strength and ease of use with a 3-axis CNC machine. The timber could be fire-treated before cutting commenced, greatly simplifying the process. By designing with just a 3-axis machine in mind, fabrication costs were significantly lower as well as reducing the amount of information required by the fabricator from the design team.

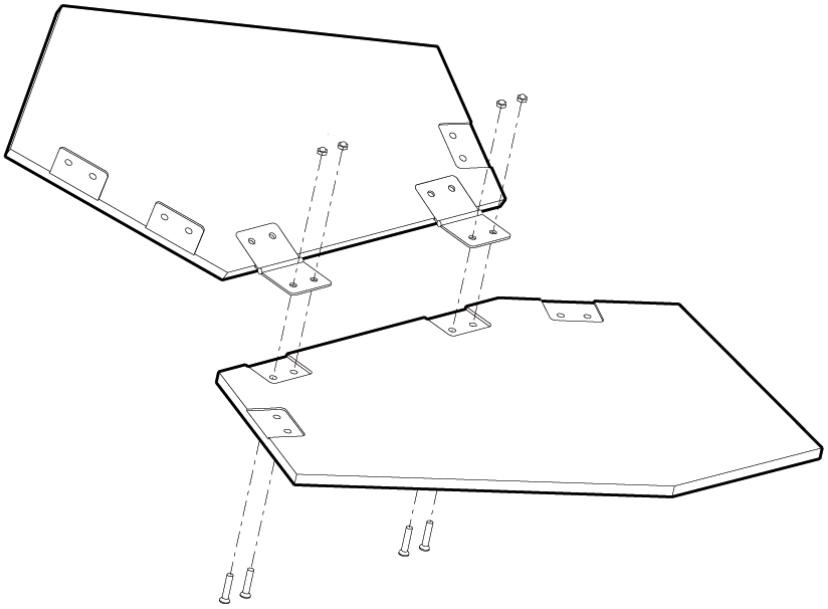


Figure E.7: Identical hinges were used to connect every panel

E.4.1. Connection Detail

As already discussed, the nature of the 3-valent shell discretisation provides sufficient geometric restraint to allow a completely hinged connection to be used, thus reducing the costs required in fabricating bespoke connections. Adjacent timber plates were therefore joined using a standard stainless steel hinge connection along each edge (fig.E.7).

Although the angle between every pair of plates is different throughout the entire structure, the hinge could easily adapt to suit both the positive and negatively curved areas. Recesses to locate the hinges were milled into each plate with each hinge then fixed to each by using a s/s M4 bolt countersunk on the outside face of the shell (fig.E.8).

E.4.2. Assembly

The final pavilion consisted of 152 panels, 900 hinges and 3600 bolts. The entire shell was flat packed and delivered to each site on two 2.4m x 1.2m x 0.9m palettes, and takes a single day to be assembled by 4 people. The speed of assembly is mainly due to the fact that no setting out information other than the support locations is required. Each panel contains a numerical reference, and so as each piece is connected the form begins to emerge as a natural consequence of the doubly curved shape and its discretisation.

Unlike concrete and masonry shells, no formwork was required. Instead, vertical props to support the structure were proposed, although at this

Figure E.8.: Assembled node detail

scale it was found that the fixed base supports allowed the legs to simply cantilever before they were joined. At the base supports, hinges were also used to provide a shadow gap of identical width to that between each panel in the main structure. These hinges were screwed into a plywood base in order to provide suitable horizontal restraint.

The final shell form (fig.E.9 & fig.E.10) covers 8m by 6m and is 4m tall at its highest point, governed by a planning requirement should the structure be used externally in the future.

E.5. Conclusion

The design of the TRADA Pavilion highlights how a new combination of computational design techniques can be used to incorporate structural and fabrication logic into the design of shell structures and allow different materials to be considered in realising such forms. The recent rise in free-form surface geometry with little structural logic has given prominence to geometric post-rationalisation methods. However, when similar techniques are applied to structurally efficient shell forms, perhaps new and interesting ideas with regards to realising free-form funicular shells become possible.

Figure E.9: Completed Shell at the Surface Design Awards (2013)

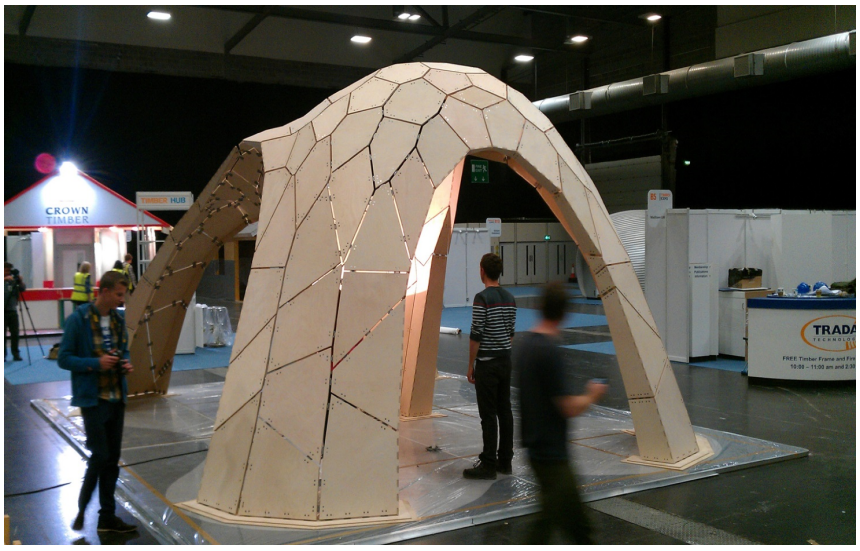


Figure E.10: Completed Shell at the UK Timber Expo (2012)

Acknowledgements

This work has been supported by Ramboll UK and the EPSRC funded Industrial Doctorate Centre in Systems at The University of Bath (Grant EP/G037353/1). Our thanks go to Kevin Atkinson for his VSA guidance early in the project.

F. Selected Algorithms

Dynamic Relaxation

A simple algorithm that simulates large displacements to a structure using a non-linear numerical method. This was used in order to improve the structural performance of models under self-weight by applying the 'hanging chain' principal in order to reduce bending moments.

Algorithm F.1 Typical dynamic relaxation algorithm

```
importData(nodes)
importData(bars)
float minKineticEnergy
%Convergence criteria
while systemEnergy < minKineticEnergy do
  for all bars b do
    %Exert forces on connected nodes based on current bar strains
    bars_b.influence(nodes)
  end for
  for all nodes n do
    nodes_n.gravity(9.81) %Apply external loads on nodes
    nodes_n.damp(0.95) %Apply viscous damping
    nodes_n.update() %Integrate positions based on residual forces
  end for
  drawSystem(nodes, bars)
  calculateTotalEnergy(systemEnergy)
end do
exportData(nodes)
exportData(bars)
```

Node Pushing

This algorithm uses a structural heuristic to *push* material towards areas of high stress and therefore improve the performance of the structure from an imported geometry.

Algorithm F.2 Node pushing to high stress areas

```
importData(nodes)
float distThreshold
bool isImproving = True
while isImproving do
  for all nodes n do
    for all otherNodes o do
      calculateDistance(nodes_n, otherNodes_o)
      if distance < distThreshold
        neighbours.add(otherNode_o)
      end if
    end for
    findStressCentroid(neighbours, stressCentroid)
    move(nodes_n, stressCentroid)
  end for
  analyseStructure(isImproving)
end do
exportData(nodes)
```

Passive Solar Design

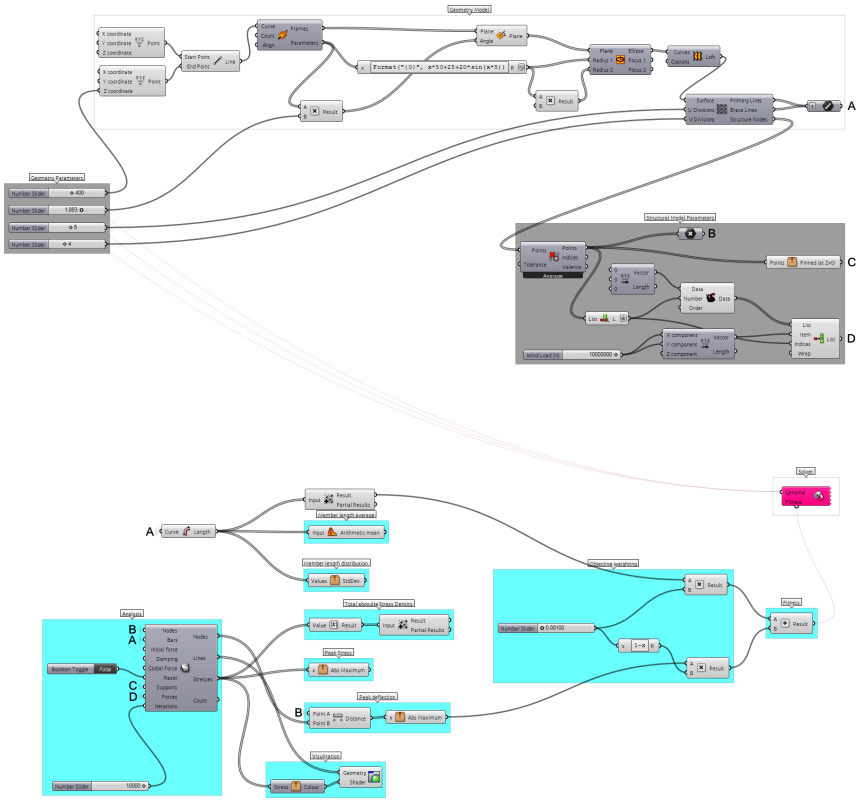
By using knowledge from our environmental engineers, a computational approach was developed that assembles a roof form based on an initial surface geometry provided by the architect and the position of the summer sun. The roof therefore provides adequate shade for summer whilst allowing the winter sun to strike the façade.

Algorithm F.3 Creating an optimal roof form using an environmental heuristic

```
importData(roofSurface)
importData(facadePanels)
float[] newNodes
for all summerDays d do
  for all peakHours h do
    for all facadePanels f do
      Point iPoint = intersectSunRay(facadePanels_n, roofSurface)
      if iPoint != null
        newNodes.add(iPoint)
      end if
    end for
  end for
end for
newRoofSurface = createRoof(newNodes)
exportData(newRoofSurface)
```

Cheongna Tower

This visual program in grasshopper is included to show how numerical parameters, geometry, structural analysis and metaheuristic solver can be included in a single graph representation.



Embryo

Embryo itself is currently around 10,000 lines of code and hence will not be included here. I have instead chosen to include the DAG generation routine, with some of the key methods excluded (they should be self-explanitory). This code has been converted from C# to Python, and then further modified for clarity.

```
/***************************************************
* Embryo: DAG generator for Grasshopper
* Copyright (c) 2013 John Harding
* class EmbryoMain : Grasshopper Standard Component
***************************************************/

# Key varibles
canvas                  # The grasshopper canvas
myComponents            # List of Embryo components
mySliders               # List of Embryo sliders
outputStash             # List of all parameter outputs
componentGUIDs          # List of ingredient ids
```

```
# Document object lists
willingObjects          # List of Parent canvas outputs
PlugObjects             # Plugged outputs on child canvas
ParentInputParams       # List of Input parameters
ParentInputComponents   # List of Input components
getGeometryComponents   # Components that collect geometry
localSettings           # Embryo settings

# Inputs from user
rMetric                 # Seed for sliders
rTopolo                 # Seed for topology/functions
sCount                  # Number of sliders to generate
cCount                  # Number of components to generate

# Default constructor
def EmbryoMain() : base()

# Solution method called by grasshopper
def SolveInstance(DA):
    # Get information from user
    getInputs(rMetric, rTopolo, sCount, cCount, localSettings)

    # Embryo works at the end of a current solution.
    mySolution = Document.SolutionEndEventHandler(DAGgenerator)
    canvas.Document.SolutionEnd += mySolution

    # Set Embryo component outputs
    setComponentOutputs(myComponents, masterCounter)

# The DAG Generator
def DAGgenerator():
    # Clear only the previously generated components
    ClearGeneratedSolution()

    # If first iteration then get information
    if (masterCounter == 0):
        ClearEverything()

        # Get all the things off the canvas
        canvasObjects = GrasshopperActiveObjects()

        # Store things based on their document object type
        foreach(canvasObjects):
            if(canvasObject is in bottom left quadrant):
                if (canvasObject is unlocked)
                    componentGUIDs.Add(canvasObject.componentGUID)
            else:
                switch(canvasObject.GrasshopperType):
                    case (Is a Parent Output):
                        willingObject.Add(canvasObject)
                    case (Is a Child OuputPlug):
                        PlugObject.Add(canvasObject)
                    case (Is a Parent Input Component):
                        canvasObject.RemoveAllSources()
                        ParentInputComponent.Add(canvasObject)
                        ParentInputParam.Add(canvasObject.Param)
                    case (Is a get geometry component):
                        ParentInputComponent.Add(canvasObject)
                # Existing components placed in child quadrant
                if(canvasObject is in top right quadrant):
                    myComponents.Add(canvasObject)

        # Raise some errors if we haven't collected enough things
        if (componentGUIDs.Count == 0 and cCount > 0):
            # Get out of here
            AddRuntimeMessage("Error 01")
```

```
        return # Get out of here
    if (myComponents.Count = 0 and cCount == 0):
        AddRuntimeMessage("Error 02")
        return # Get out of here

# Now begin to add to the output parameter stash
foreach(willingObjects) outputStash.Add(willingObject.output)

# Make the sliders, add to canvas and add to the outputstash
for(sCount):
    mySlider = new Grasshopper_NumberSlider
    mySlider.setminmax(mySettings.data)
    mySlider.SetSliderValue(rMetric.Next*domain)
    mySliders.Add(mySlider)
    Document.addObject(mySlider)
    outputStash.Add(mySlider.output)

# Make the NEW components and add them to the canvas
for (cCount):
    selection = rTopolo.Next(0, componentGUIDs.Count)
    myComponent.Add(new Component(componentGUIDs.selection))
    Document.addObject(myComponent)

    # Identify as a generated component
    myComponent.Component.Message = "G"

# Shuffle the deck
ShuffleComponents(myComponents, rTopolo.Next)

# Wire up each input
foreach(myComponents):
    # Avoid hooking up to same output
    stashRecord = new List

    foreach(myComponent.Inputs):
        WireUp(myComponent.Input, rTopolo, stashRecord, isCD)

    # Get rid of the component if it failed.
    if (myComponent failed to collect data):
        myComponent.Alive = false
        if (mySettings.RemoveDead):
            Document.RemoveObject(myComponent)
    else:
        myComponent.Component.Hidden = mySettings.Preview
        foreach(getGeometryComponents):
            geometryOut(getGeometryComponents[i])

        # Add the outputs to the stash
        foreach(myComponent.Outputs):
            if(plugObjects.contain(myComponent.Output)
                outputStash.Add(myComponent.Output)

# Finally, connect any parent inputs
ShuffleObjects(outputStash, rTopolo)

foreach(ParentInputParams):
    WireUp(null, rTopolo, stashRecord, isCD)
    myObjects = ParentInputComponent.AllDownstreamObjects()
    foreach (myObjects)
        myObject.ComputeData()

# Perform topological sort and
# add to canvas in dependency order for easier cognition
TopSortComponents(myComponents, mySettings.Cartesian)

# Iterate the counter
masterCounter++
```

```
# Method which hooks up an input to an existing output
def WireUp(thisInput, rTopolo, stashRecord, isCD):
    flag = false
    s = 0
    Friends.ShuffleObjects(outputStash, mySeed)
    while (!flag)
        # Parameter type
        # TODO: Perhaps use volatile data type instead
        myType = thisInput.ParameterType()
        yourType = outputStash[s].ParameterType()

        if(myType is a Slider):
            if (SliderTypeCheck.isValid(myType, yourType)
                and !stashRecord.Contains(outputStash[s])):
                thisInput.AddSource(outputStash[s])
                flag = true
                if (isCD): outputStash.RemoveAt(s)
                else: stashRecord.Add(outputStash[s])

        else if (myType is a Component):
            if (ComponentTypeCheck.isValid(myType, yourType)
                and !stashRecord.Contains(outputStash[s])):
                thisInput.AddSource(outputStash[s])
                flag = true
                if (thisInput != null):
                    foreach (myComponents):
                        foreach(myComponent.Outputs):
                            if (myComponent.Outputs.Equals(outputStash[s]))
                                thisInput.AddInput(myComponent.Output)
                                if (mySettings.OnlyTerms):
                                    myComponent.Hidden = true
                if (isCD): outputStash.RemoveAt(s)
                else: stashRecord.Add(outputStash[s])

        # If we've tried everything, then give up
        if (!flag and s == outputStash.Count - 1):
            # Indicate failure to the user
            if (thisComponent != null) thisComponent.Message += "X"
                flag = true

        # Else keep trying...
        else:
            s++
```

G. Embryo Examples

A series of generated Grasshopper definitions using four primitive components: Point by xyz, Box by 2 points, Line by 2 points, Divide curve.

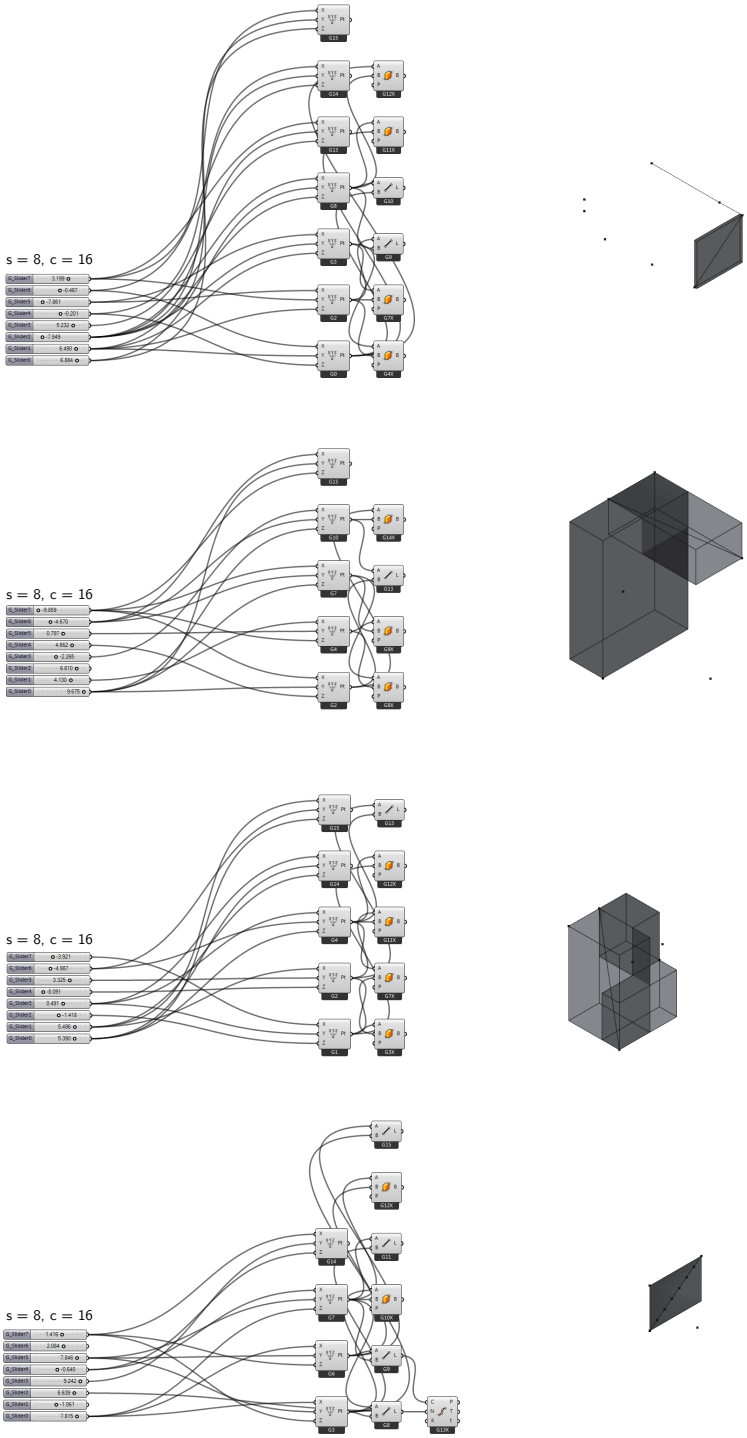


Figure G.1: Different examples for 8 sliders and 16 components

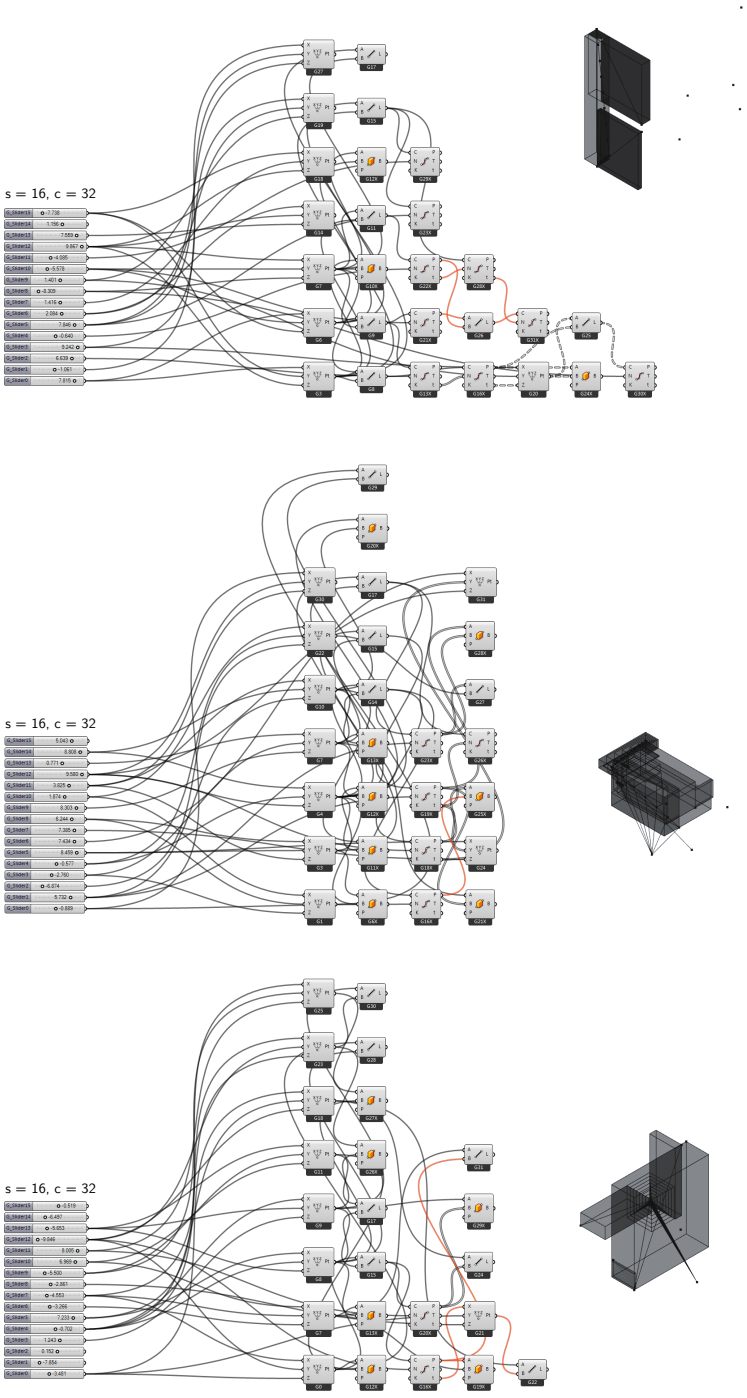


Figure G.2: Different examples for 16 sliders and 32 components

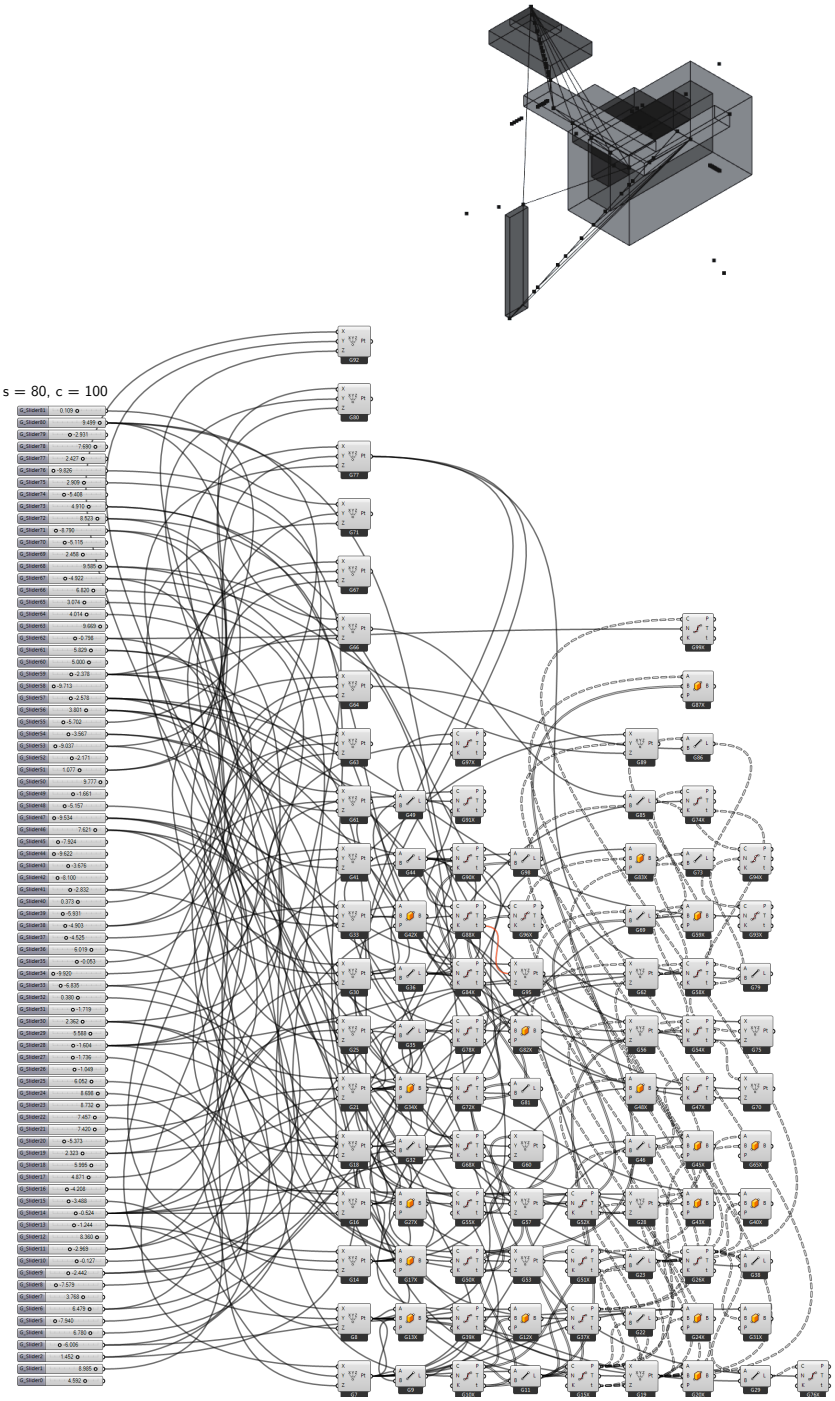


Figure G.3: Examples for 80 sliders and 100 components

H. More Embryo Examples

H.1. Tower Hamlets: Embryo Setup

H.2. Tower Hamlets: Examples

H.3. Tower Hamlets: Graph Edit

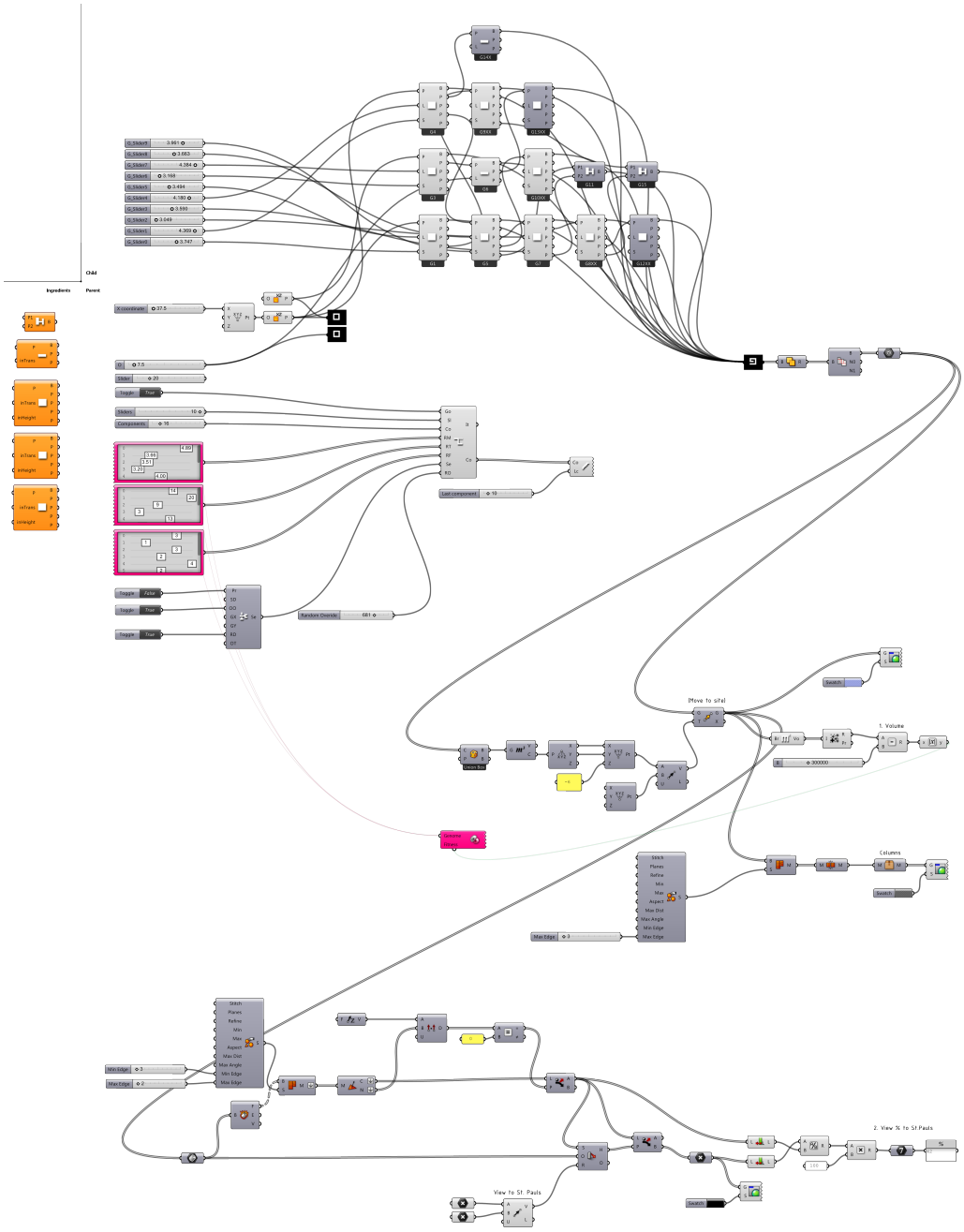


Figure H.1: Grasshopper/Embryo setup on the Tower Hamlets residential project [33Tow].

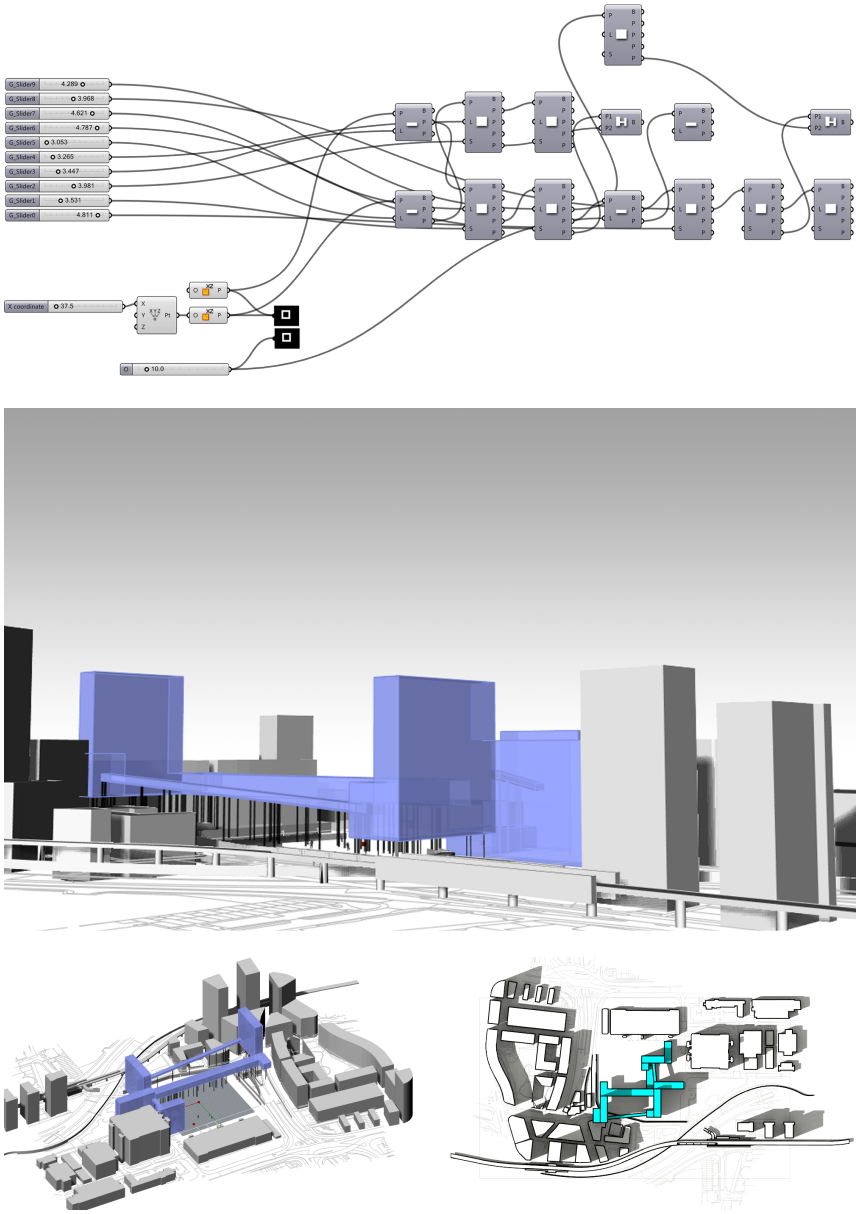


Figure H.2: Design 'B'

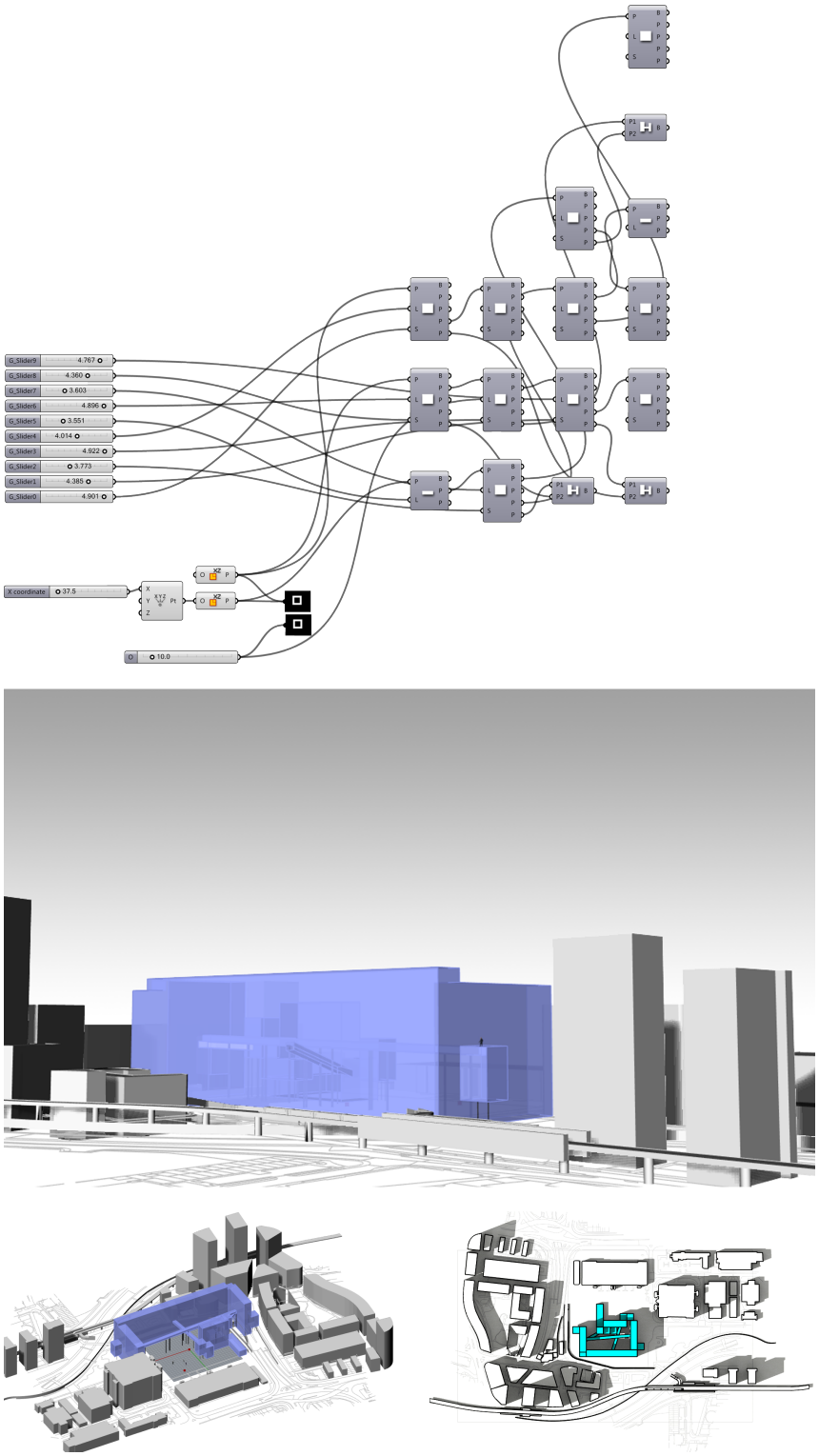


Figure H.3: Design 'C'

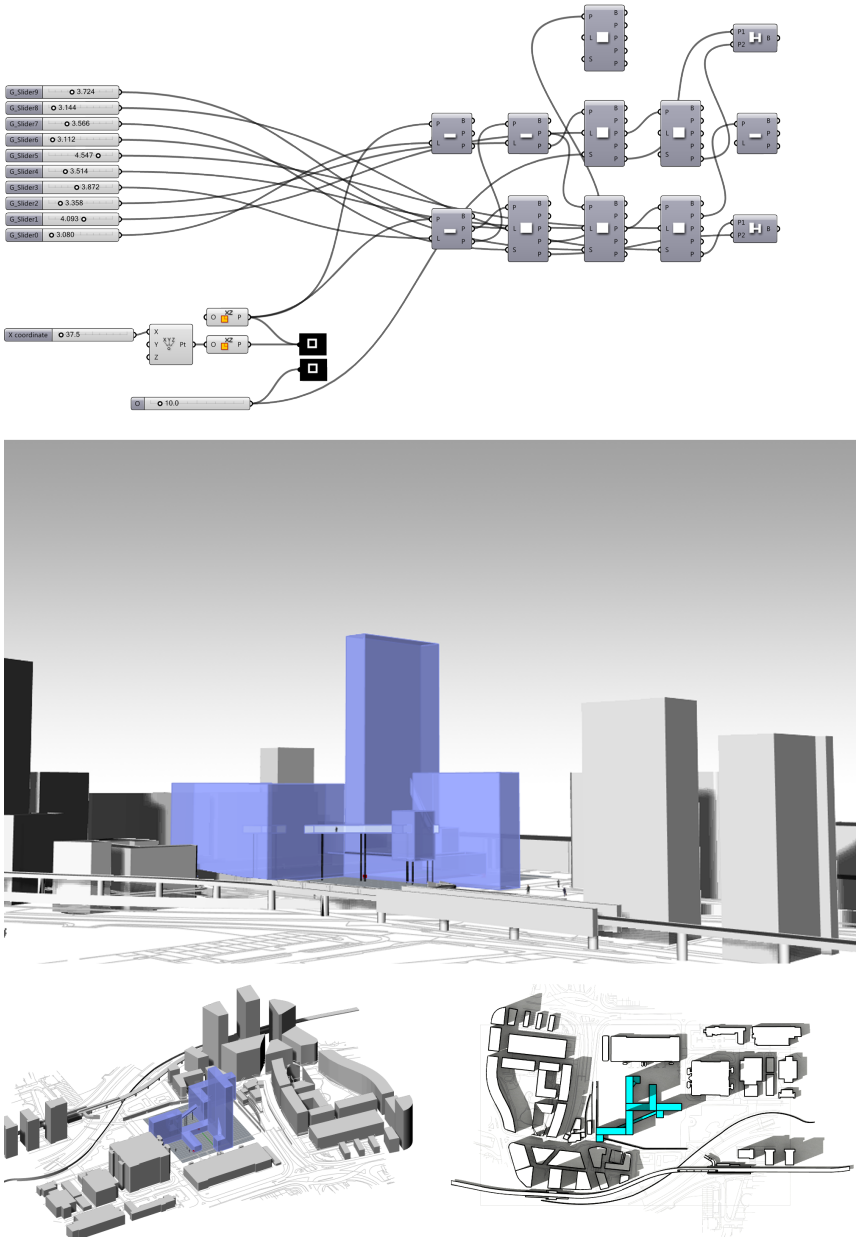


Figure H.4: Design 'D'

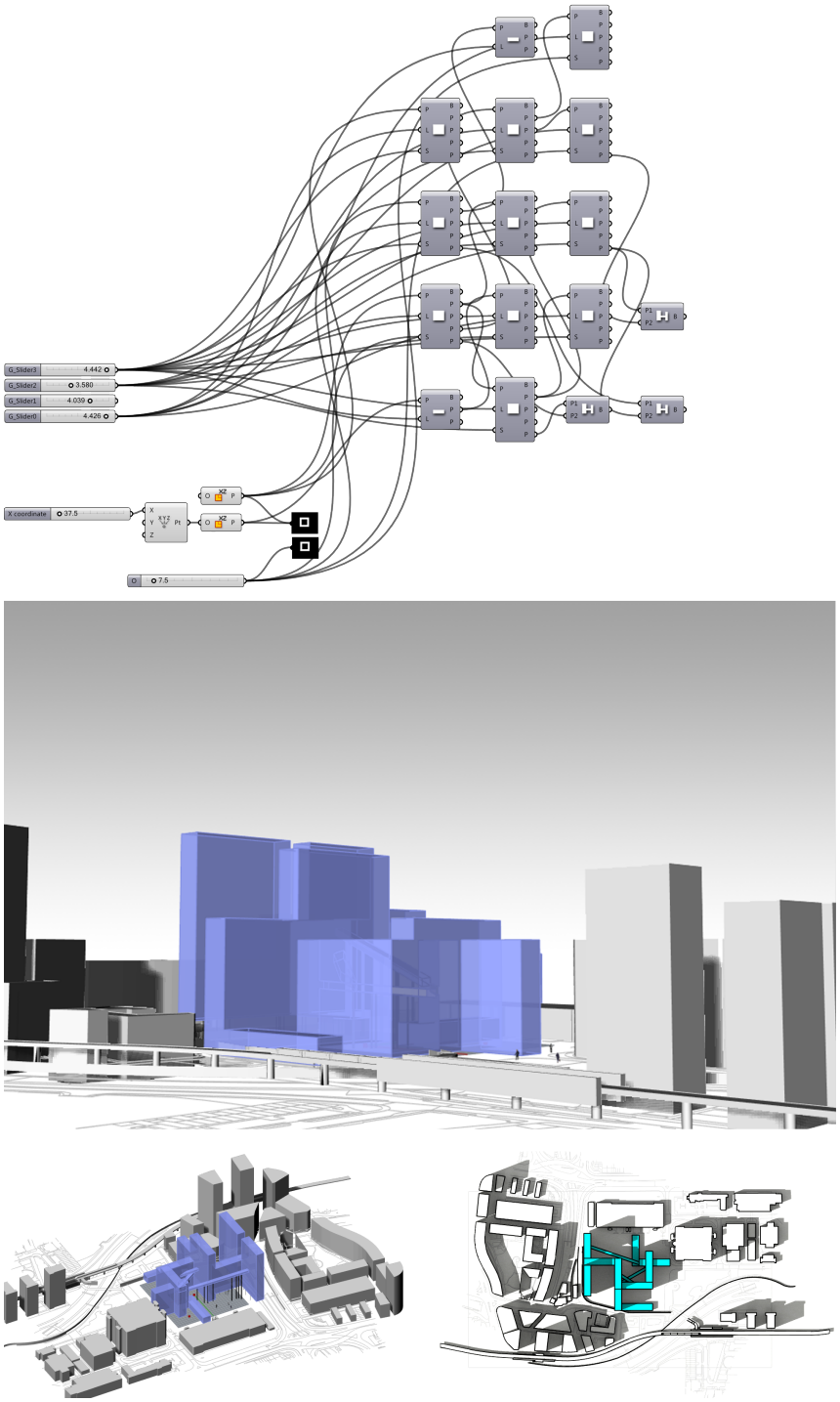


Figure H.5: Design 'E'

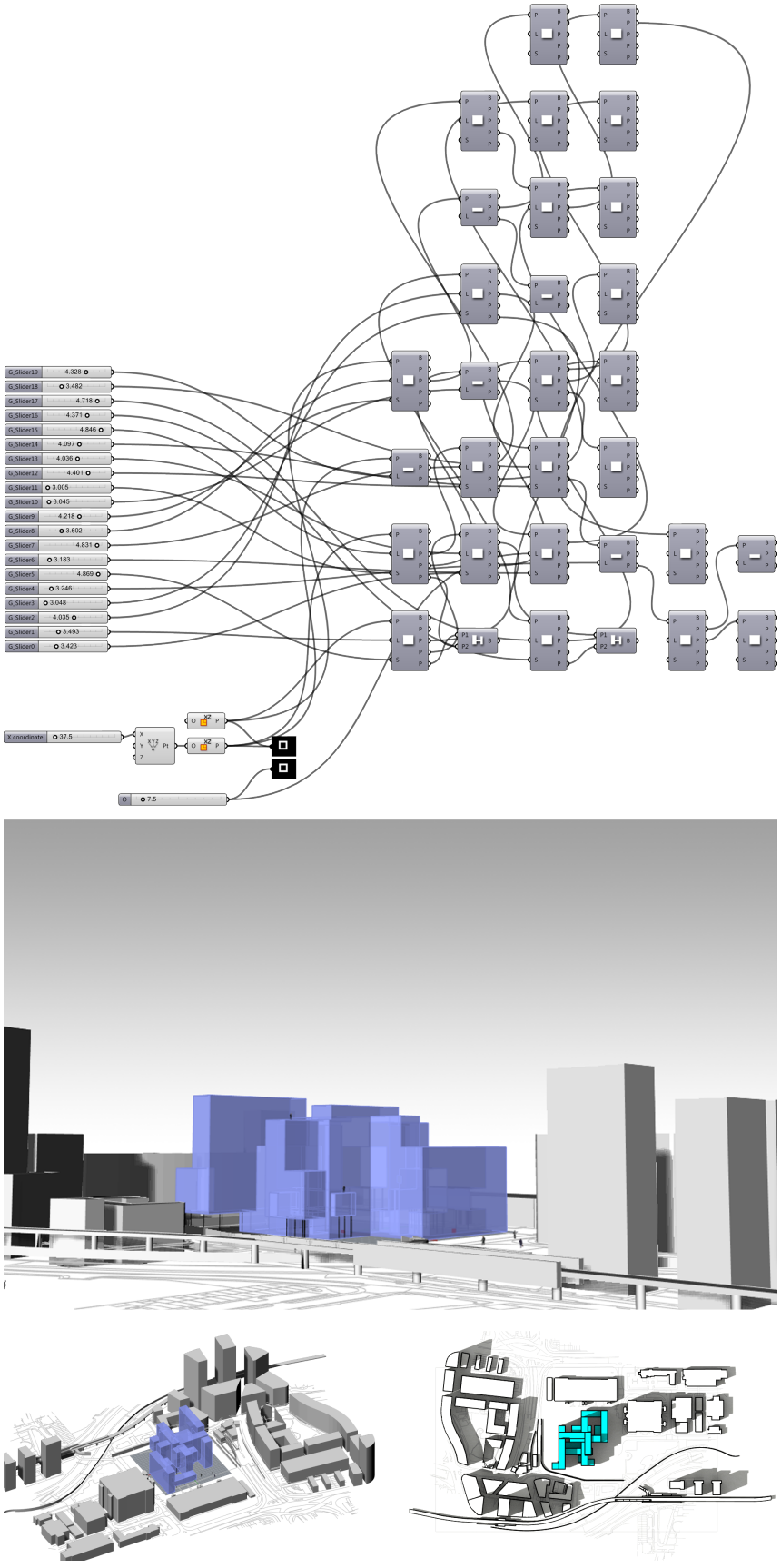


Figure H.6: Design 'F'

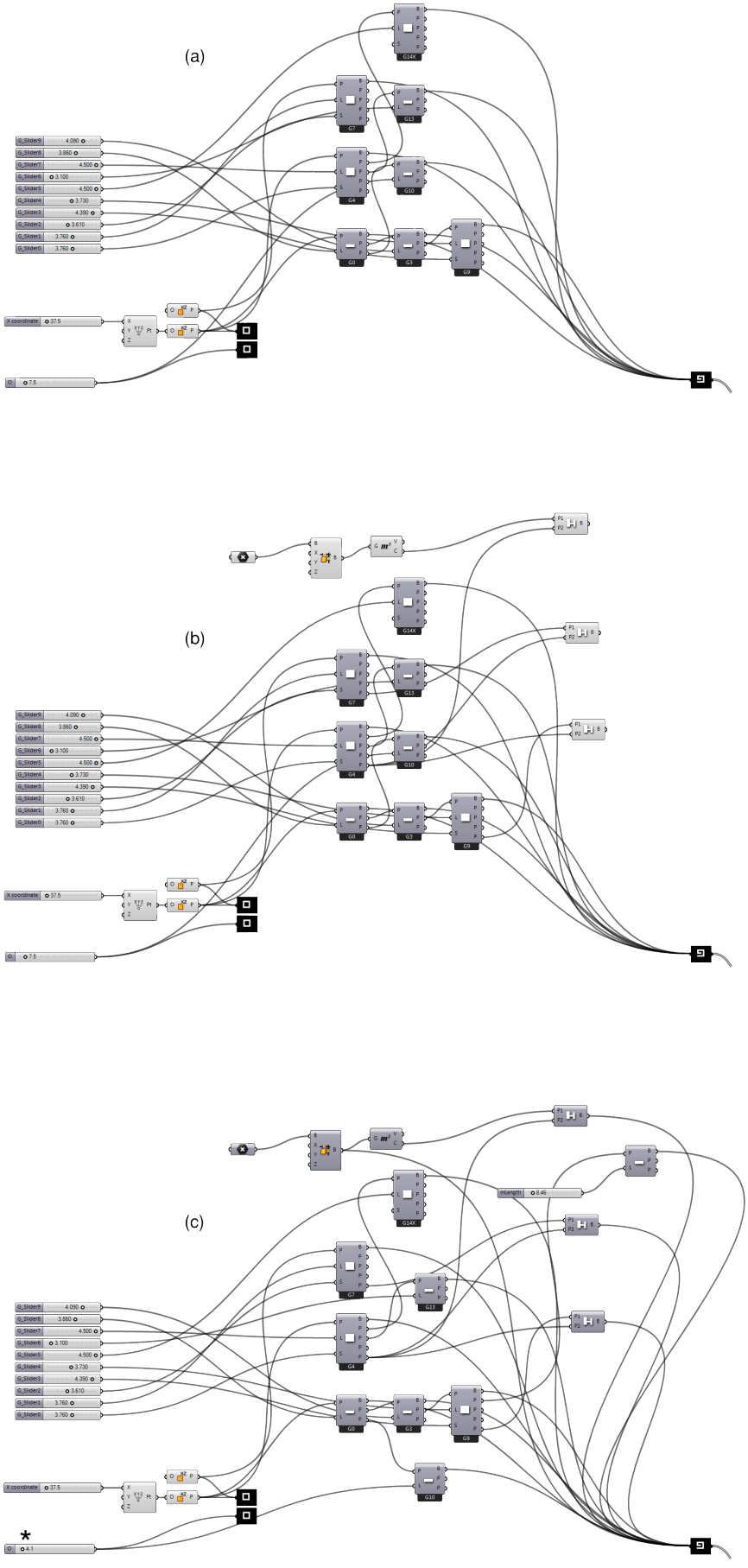


Figure H.7: Graph Development post-Embryo (1/2)

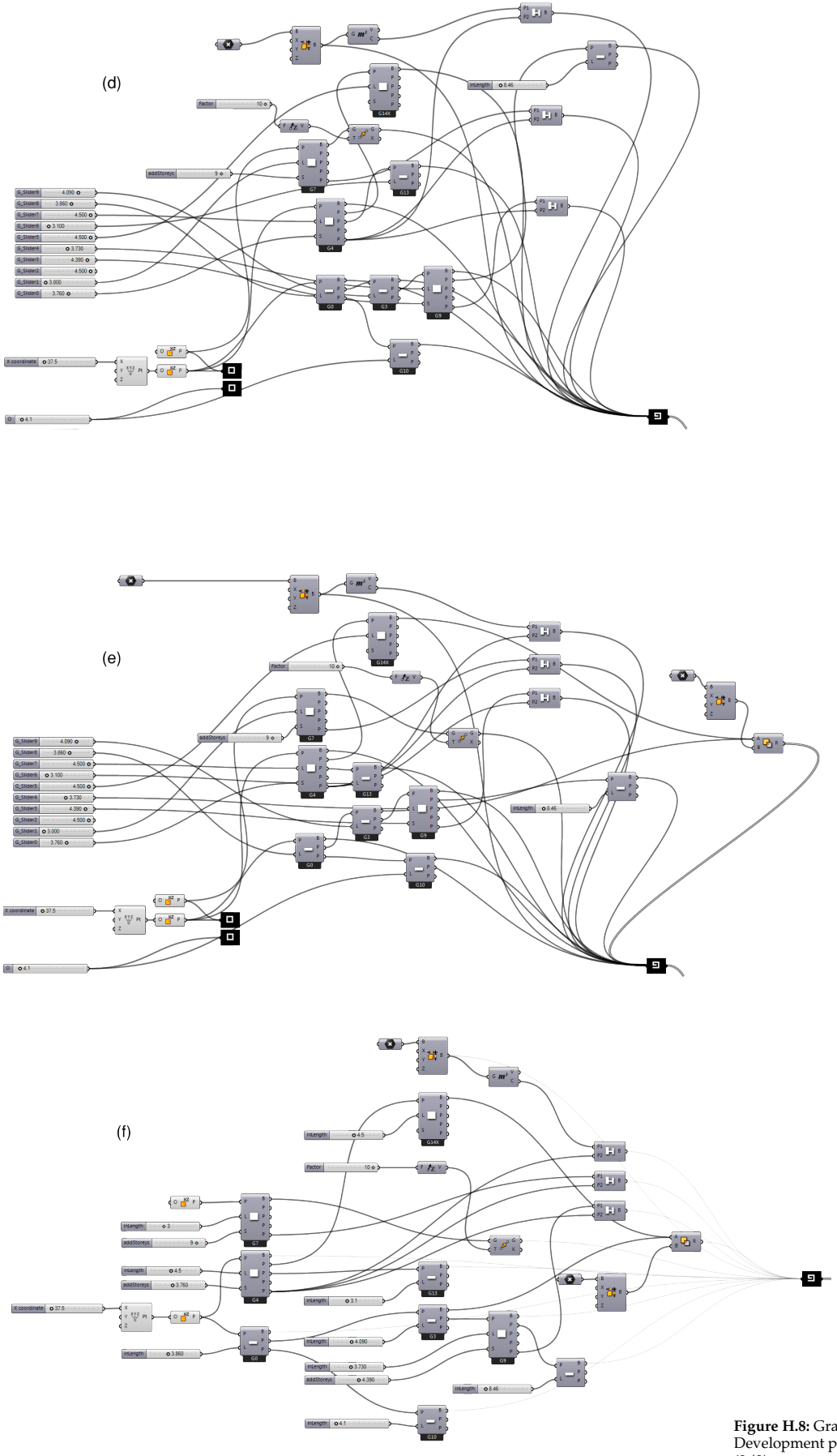


Figure H.8: Graph Development post-Embryo (2/2)